

**F/G 9/2**

MARTIN MARIETTA AEROSPACE DENVER CO DENVER DIV  
TOTAL SYSTEM DESIGN METHODOLOGY. (U)

**F30602-78-C-0250**

**UNCLASSIFIED**

RADC-TR-80-337

NL

1061  
409: 127

END  
DATE  
FILMED  
3-8  
DTIC

AD A 095727

✓  
**RADC-TR-80-337**  
Final Technical Report  
January 1981

**LEVEL**

**II**



12

# **TOTAL SYSTEM DESIGN METHODOLOGY**

**MARTIN MARIETTA AEROSPACE**

**EDWARD C. STANKE, II**

**DTIC**  
**ELECTE**  
**S D**  
**MAR 02 1981**  
**E**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DBC FILE COPY

**ROME AIR DEVELOPMENT CENTER**  
**Air Force Systems Command**  
**Griffiss Air Force Base, New York 13441**

01 0 2 1 2 2

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

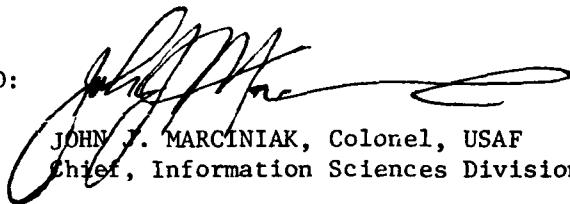
RADC-TR-80-337 has been reviewed and is approved for publication.

APPROVED:



NATHAN B. CLARK, Capt, USAF  
Project Engineer

APPROVED:



JOHN J. MARCINIAK, Colonel, USAF  
Chief, Information Sciences Division

FOR THE COMMANDER:



JOHN P. HUSS  
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCA), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-80-337	2. GOVT ACCESSION NO. AD-A045 72 7L	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) TOTAL SYSTEM DESIGN METHODOLOGY	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report, 23 Aug 78 - 31 Aug 80.	
7. AUTHOR(s) Edward C. Stanke, II	6. PERFORMING ORG. REPORT NUMBER N/A	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Martin Marietta Aerospace (Denver Division) PO Box 179 Denver CO 80201	8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0250	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCA) Griffiss AFB NY 13441	10. PROGRAM ELEMENT PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55811704	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	12. REPORT DATE January 1981	
	13. NUMBER OF PAGES 64	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)  Same		
18. SUPPLEMENTARY NOTES  RADC Project Engineer: Nathan B. Clark, Capt, USAF (ISCA)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) System Design Methodology Requirements Analysis Functional Decomposition		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes the Total Systems Design Methodology, including the philosophy, automated and manual tools and procedures which support it. It also documents the applications of that methodology to the Navstar Global Positioning System Operational Control Segment system definition and conclusions based on that application.		

DD FORM 1473

1 JAN 73

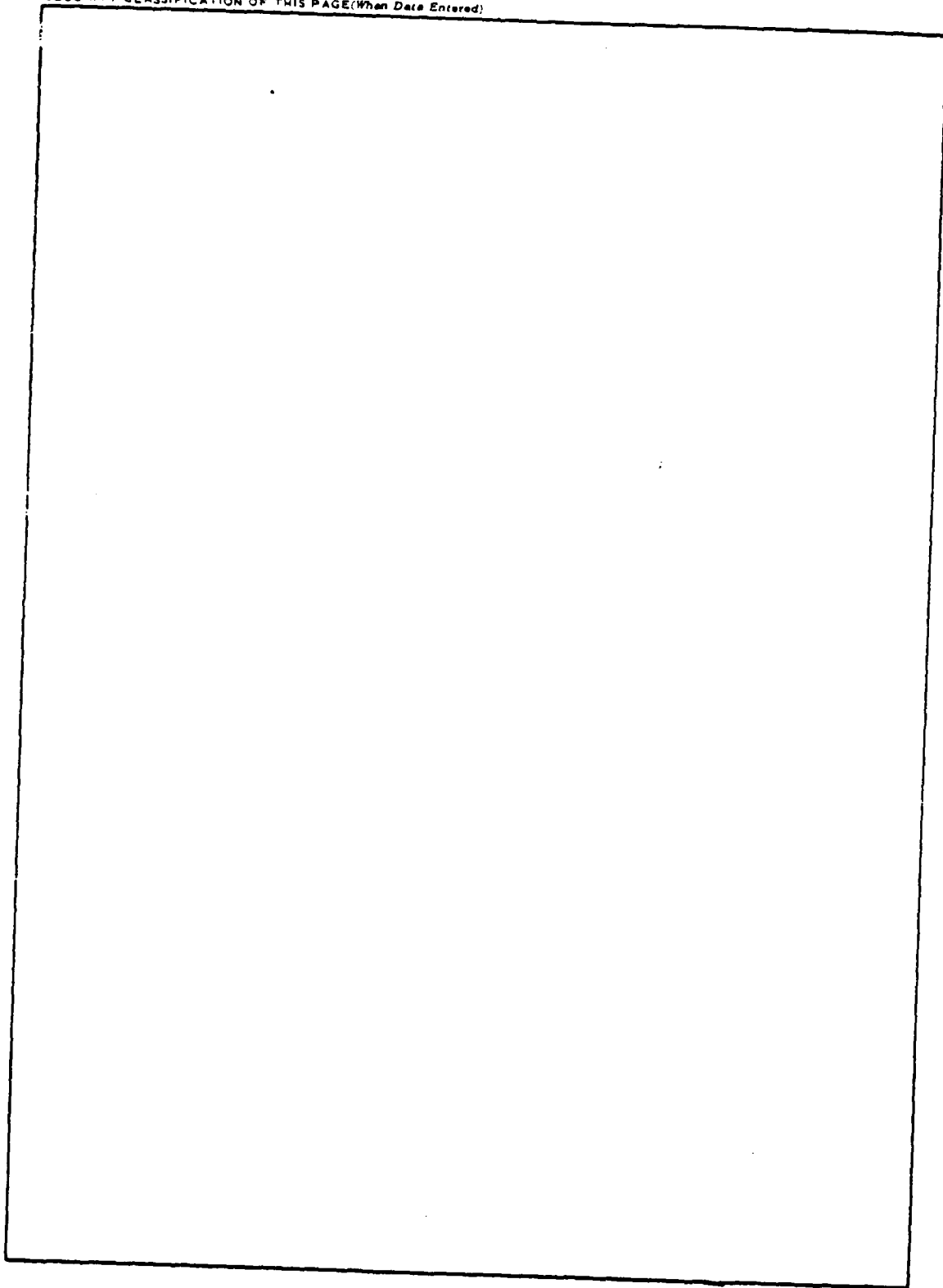
EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

# TABLE OF CONTENTS

	<u>Page</u>
1.0 INTRODUCTION .....	1
2.0 TSD OVERVIEW .....	2
2.1 TSD Conceptual Foundation.....	2
2.2 TSD Approach .....	7
3.0 TSD TOOLS DESCRIPTION .....	10
3.1 Introduction .....	10
3.2 Multi-Level Expression Design System (MED Sys).....	13
3.2.1 MEDL-R .....	14
3.2.2 MEDL-D .....	21
3.2.3 MEDL-P, MEDL-X .....	22
3.2.4 GAPS .....	24
3.3 Comprehensive Computer Network Modeling (CCNM).....	25
3.4 Non-Automated Tools/Procedures .....	26
3.4.1 Data Flow Diagrams .....	27
3.4.2 N <sup>2</sup> Charts .....	34
3.5 Other Tools.....	35
4.0 TSD USAGE.....	42
4.1 Introduction .....	42
4.2 Proposal .....	42
4.3 Functional Baseline.....	45
4.4 Allocated Baseline.....	48
5.0 GPS USE OF TSD .....	51
5.1 Introduction .....	51
5.2 Approach.....	51
5.3 Results.....	54
5.4 Conclusions .....	55
6.0 CONCLUSIONS .....	57

Accession For		
NTIS GTR&I	<input checked="" type="checkbox"/>	
DTIC TAB	<input type="checkbox"/>	
Unannounced	<input type="checkbox"/>	
Justification		
By		
Distribution/		
Availability Codes		
Dist and/or		
Special		
A		

## EVALUATION

This effort has advanced the state of the art in computer system design methodologies by initiating the implementation of a TSD methodology originally developed by Martin Marietta in their IR&D program. This effort also provides a case study analysis as a method of demonstrating current capabilities and areas for further implementation.

A handwritten signature in cursive script, reading "Nathan B. Clark".

NATHAN B. CLARK, Capt, USAF  
Project Engineer

## 1.0 Technical Problem

The Total System Design (TSD) Methodology contract had as its goal the documentation of Martin Marietta's approach to embedded computer system design and the documentation of the application of that approach to a real world program. The problem addressed by TSD is the wide variability of projects involving computers in terms of system success. The intent of the methodology is to impose a rational approach, supported by automated and manual tools and procedures, on the process of defining the computer portion of systems. The computer portion includes the basic hardware, the software driving the hardware and the people who interact with the hardware and software.

## 2.0 General Methodology

The methodology used in the contract stems from the two purposes of the contract. First is the documentation of the TSD methodology, including philosophy and supporting tools, both automated and manual. The second is the documentation of the results from applying the TSD methodology to the Navstar GPS Operational Control Segment. The contract under which the GPS work was defined was a Stage 1 contract whose output was A and B level specifications of the Operational Control Segment of the GPS system. This segment represents a command and control system with aspects of real time, distributed processing.

## 3.0 Technical Results

The application of the TSD methodology to the Navstar GPS project provided several significant conclusions:

1. The requirements analysis process must be aided by some sort of automated tool. The benefits received will more than compensate



for the initial front end costs.

2. The use of simulation for verification purposes is absolutely necessary for the architecture definition phase.
3. The step of going from a complete understanding of the functions of the system to the actual system implementation components is one serious gap in the methodology.
4. The use of TSD requires an understanding of the methodology and belief of the people using it in its ability to aid the process.

The overall assessment of the methodology is that it provides a useful frame of reference and a viable approach which aids system design, but the methodology is incomplete and more research needs to be done in the area of system design based on functional decomposition.

## 1.0 INTRODUCTION

This report documents one view of the design process for computer systems and the application of that process to a real world problem. The use of the word system in this report refers to the computer portion of a larger system, which may in general be more encompassing than just the computer. For example, the sensors in a surveillance system are very much a part of the system itself but are not included in the definition of system within this report.

The intent of the Total System Design (TSD) Methodology is to impose a rational approach to defining what the computer portion of a system should be, including the underlying hardware, the software which drives the hardware and the people who interact with the hardware and software. As currently implemented, the methodology does provide a framework for that rational approach. There are a number of tools and procedures supporting the methodology. However, there are gaps and the interfaces between the various tools are not always well defined. Work at Martin Marietta in the area of methodology and tools is on-going. This work supports the filling of the perceived gaps and the development of a consolidated support system for system development.

This report details the current philosophy and implementation of the methodology. Section 2.0 gives an overview of TSD by providing a conceptual foundation and approach. Section 3.0 describes some of the tools currently supporting TSD and Section 4.0 gives a scenario of TSD use in a system design. Section 5.0 is the description of TSD application to a real world problem, the Global Positioning System (GPS). Finally, Section 6.0 describes where the methodology is now and gives a critique of the methodology based on actual use.

## 2.0 TSD OVERVIEW

### 2.1 TSD Conceptual Foundation

The category of command, control and communications ( $C^3$ ) systems encompasses a broad spectrum of military and scientific worlds. Examples of  $C^3$  systems applications include military tactical and strategic command centers, military intelligence networks, process control systems, satellite operations centers and launch control facilities. In the past, some  $C^3$  systems have performed successfully and others have failed miserably. The wide disparity in the results of the system development, together with the extremely high cost of failure, have created an urgent need to reduce or eliminate faulty system design for future systems.

Initial efforts in trying to cope with increasing system complexity emerged from researchers in the software arena and resulted in concepts such as structured programming and top-down design. The "Software First Concept" has emerged relatively recently in response to the continuing decrease in hardware prices and recognition of the fact that software costs can be, and very often are, the overriding factor in total  $C^3$  system cost. "Software First" implies that the system designer can decide how to most effectively implement the system's functional requirements and then select the appropriate hardware, as opposed to having the software being categorically driven as a result of premature specification of hardware.

The system development process stands to benefit from this software activity. Top-down design concepts, for example, are just as applicable to the process of defining the hardware, software and the action of people, referred to collectively as the system, as they are to software design. Likewise, the tools developed primarily for software (e.g., requirements tools) have a wider applicability.

Since many of these tools are automated, the possibility arises of developing a computer-aided design system to aid the system design process. Computer-aided system design is appealing because conceptually state-of-the-art system design methods could be maintained in the computer system and accessed interactively to help a system designer in his effort to design a new system or to modify an existing system. System design could conceivably become more of a discipline and less of a subjective art. The observation can be made that C<sup>3</sup> systems in themselves are extremely varied in complexity and function and that an automated method of designing such systems would of necessity be perhaps an order of magnitude more complex than potential target systems. This may indeed prove to be the case. Nevertheless, such automated methods are in themselves systems, and as such can be implemented by the very methodology they are trying to perpetuate.

Martin Marietta is developing a "Computer System Development Methodology," a Total System Design (TSD) Methodology. TSD Methodology is a set of skills, tools, techniques and facilities whose ultimate purpose is to provide for an orderly, systematic approach to system design including the key concepts of baselining and validation at important points in the design process.

The foundation for any system design is a set of requirements which define what the system is to do. The set of requirements for the TSD methodology system can be overviewed as follows:

1. Must be receptive to system requirements, both functional and non-functional. Non-functional system requirements include software maintainability, system life-cycle costs and training. The functional requirements are stated as machine processable problem statements; more about this topic later.

2. Must be receptive to system architecture, where "system architecture" refers to the mixture of hardware, software and human functions as they interact in the functional system.

3. Must be able to compare various system designs; this requirement is obvious.

4. Must be capable of interactive design, including feedback regarding the effect of system architecture changes on the overall performance of the system.

5. Must be receptive to system requirements changes. System requirements changes must be anticipated in any major system, as a general rule. Some classic causes of system requirements changes include poor planning, incomplete or inconsistent requirements, lack of visibility and errors not detected early in the system design. In addition to these glaring causes of changes, legitimate functional changes should be anticipated due to improved technology, system expansion and new requirements.

6. Must be capable of supporting system models. The methodology includes the development and refinement of models of the target system for analysis and validation purposes. The first model produced, the system functional model, which is a model of what the system does and not how it does it and which is hence implementation independent, requires support. The more detailed models, including simulations and emulations of the target system also require support.

The processes involved in creating, analyzing and modifying the system baselines form the system development methodology. There are three basic phases in system development. These phases are interdependent but typical system development follows the basic sequence shown in Figure 1.



**Figure 1 System Development Phases**

It is important to note that the three phases described do not include the whole system life cycle, particularly the operations and maintenance of the system after delivery. The total system development methodology, by its very nature, does not include specific operation and maintenance but should, through its effects on the design, produce systems which are more easily operated, maintained and modified.

There are a number of tools, concepts and methods supporting the implementation phase. For software, for example, these include all the currently emerging software engineering concepts including top-down design, implementation and testing, structured coding, program management concepts such as chief programmer teams, structured walkthroughs and support systems such as program support libraries, program description languages and high order languages. For hardware, there are the traditional tools of simulation prototyping and breadboards. The legacy and capabilities represented by these tools, methods and concepts indicate that if the preliminary work done in the analysis and synthesis phase is viable, the implementation phase, although still a major part of the development, is more reasonably accomplished. The TSD methodology therefore concentrates on the initial two phases of the system development process, analysis and synthesis. If the initial, top-level

system design is based on rational, well-defined steps, there is a strong probability that the implementation will proceed much more smoothly and the final resultant system will stand a much better chance of not only solving a problem but of solving the right problem, the one the customer or user needs solved.

The first two phases of system development are intimately involved with the top-level system requirements. These requirements fall into two general categories: functional requirements, which tell what the system is to do, and performance requirements, which tell within what constraints the system must do it. There is a third class, which could be called general, relating to overall constraints such as power maximums, size constraints, etc. This class relates to neither function nor performance, although it has an effect on both and must be included at some point into the process.

The first phase of system development, assuming conceptual and feasibility studies have demonstrated that solution to the problem is possible and practical, is the analysis phase. During this phase, the primary emphasis is on getting a valid appreciation of the problem being solved. This appreciation includes an understanding of the various pieces necessary from a functional point of view. The major result of this phase is a functional model of the system, showing how the various components interact and the interfaces between them.

The analysis phase deals with the functional requirements for the system. Performance or general constraints are not included because they relate to how the system accomplishes its tasks, not to what the system does. The partitioning of the what and the how is fundamental to the system development process. Until the functions that the system must perform are well understood,

it is premature to even consider the resources necessary for the system. It is here that many system developments fall down. One of the primary causes of problems is the premature selection of components (computer hardware) in an effort to "show progress" on the system. The more practical approach of letting the functions to be performed dictate the system resources requires a thorough understanding of the functions before deciding what those resources are to be.

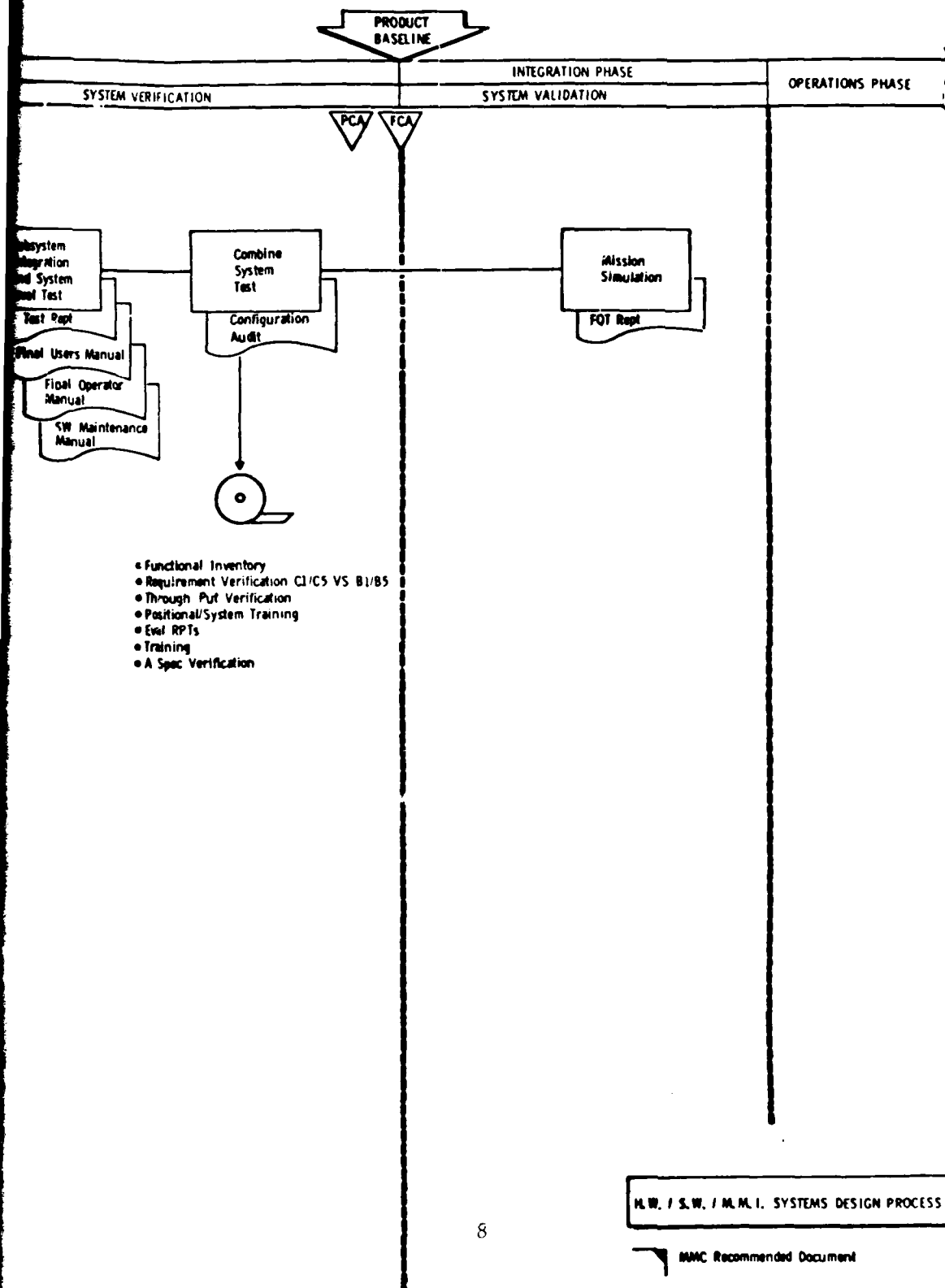
The second phase, synthesis, uses the functional model of the system developed in the analysis phase as a baseline. During this phase, the performance requirements (constraints), general requirements and current state-of-the-art are added into the next baseline of the system. This phase includes trade-off studies, simulation/emulation and whatever other analytical or experimental techniques will aid in determining the proper hardware, software and human resources to best fulfill the functional requirements within the performance and other constraints.

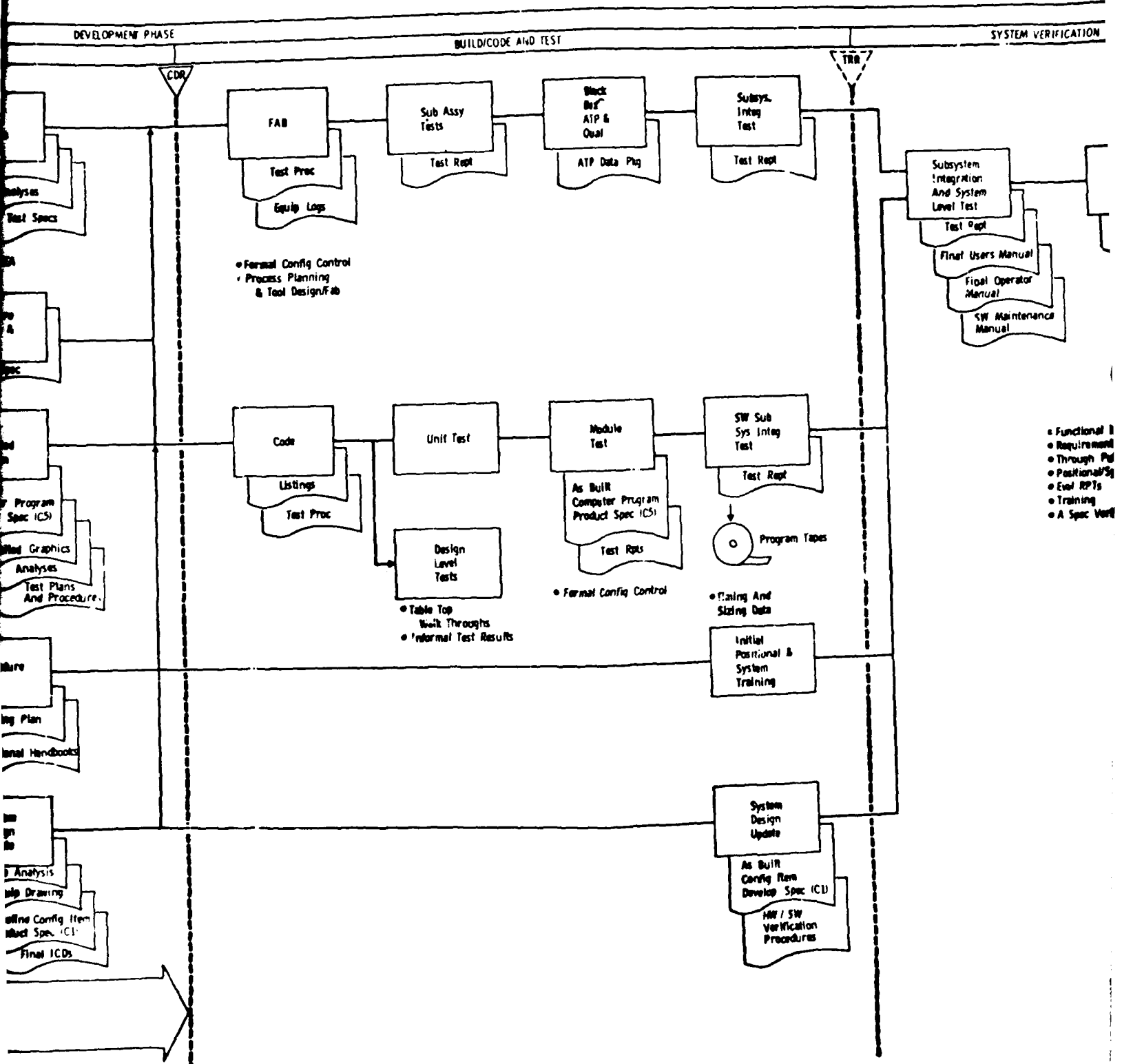
## 2.2 TSD Approach

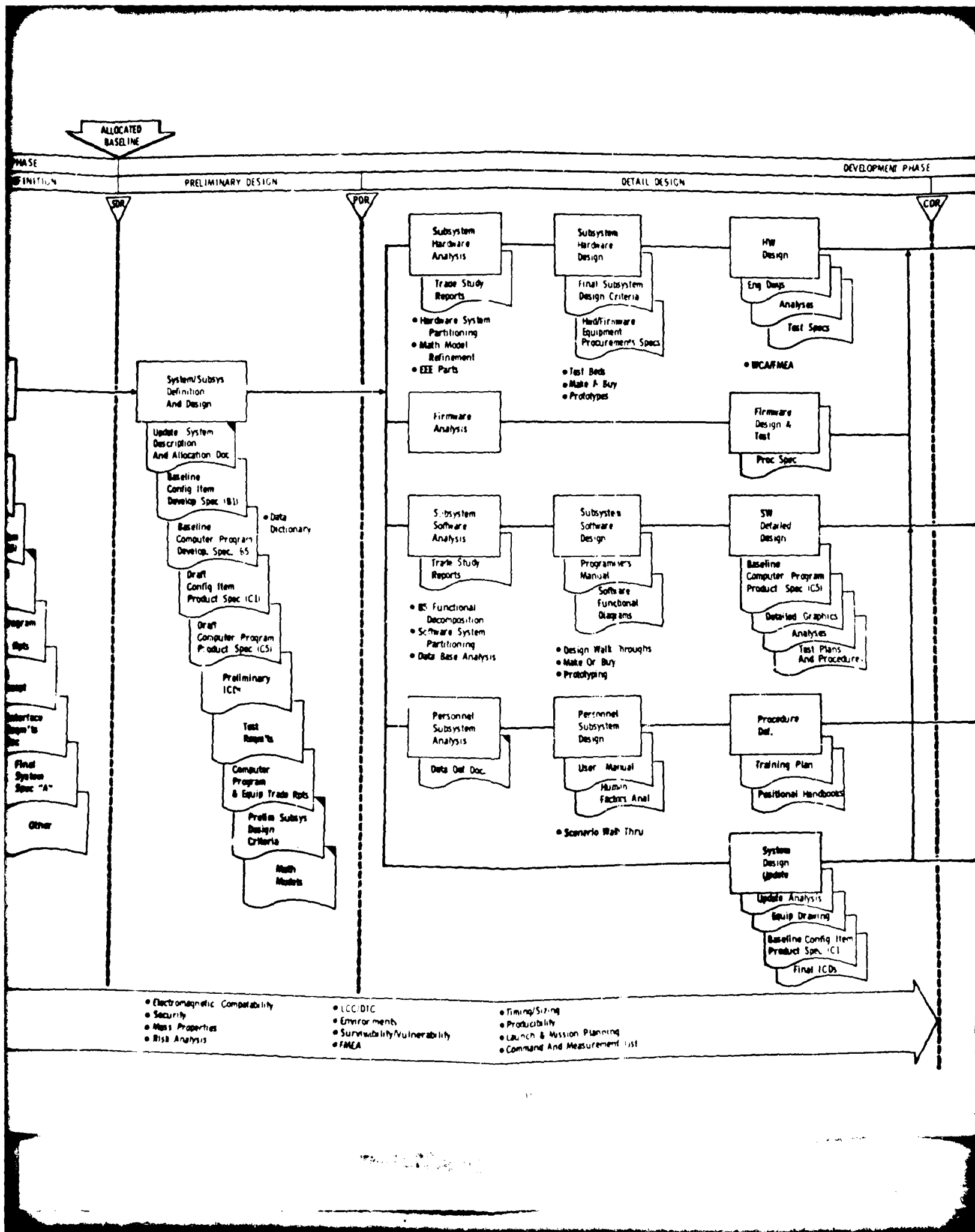
Figure 2 depicts Martin Marietta's TSD approach. Figure 2 shows the various steps which comprise the process, along with the various documents which are produced during the development. There are several significant items in this figure. First, the feedback loops are not shown. However, the realities of system development dictate that the capability to reiterate some part of the process is absolutely necessary. Second, the figure does put the stages of system design into the perspective of the reviews specified by MIL-STD-1521A.

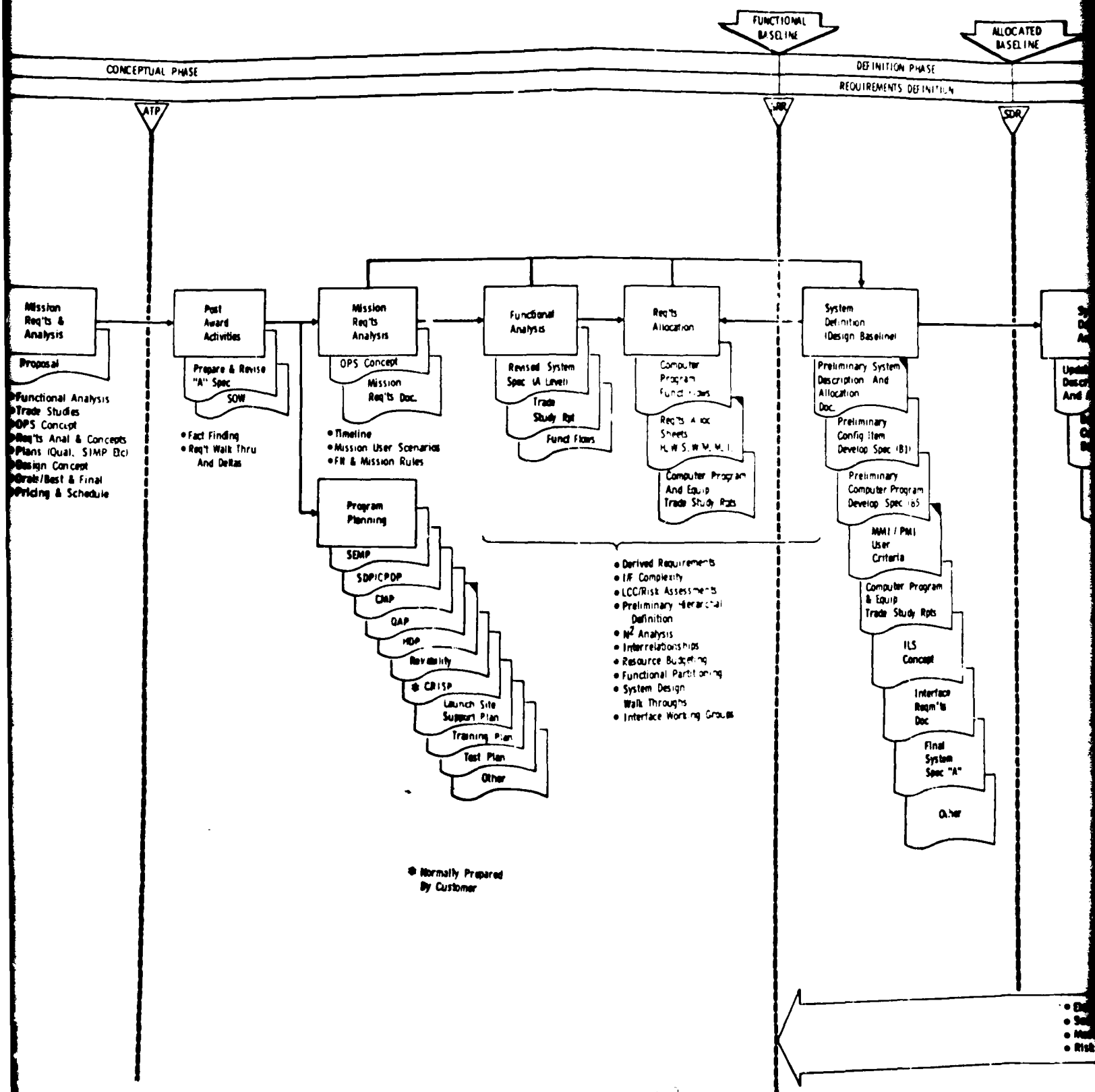
In terms of the analysis and synthesis phases described previously, everything on the chart up to the system requirements review (SRR) can be











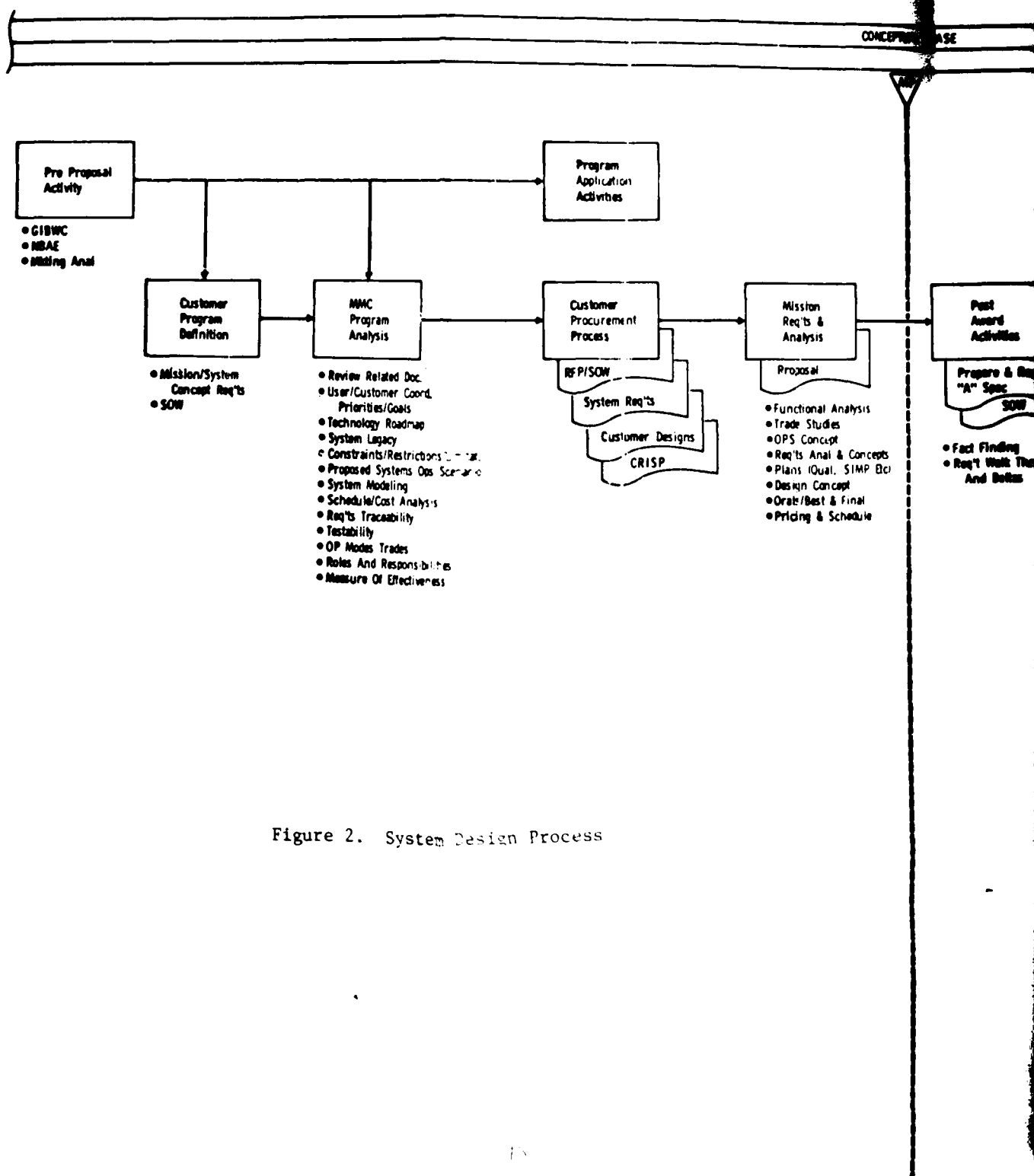


Figure 2. System Design Process

categorized as analysis. Synthesis consists of all the steps from the system requirements review to the critical design review (CDR). The balance of the figure to the operations phase represents the implementation phase.

The Martin Marietta approach to actually performing the steps of the system design process depends on a number of tools and procedures, both manual and automated. The current implementation of the methodology is not complete and does not provide the end-to-end capability envisioned as the final result. However, the concepts and philosophy which drive the methodology are in place and provide a good backdrop for tool use, evaluation and enhancement. Every effort is made to use currently available tools wherever possible to avoid "reinventing the wheel." The discussion which follows will describe the tools which are currently part of the methodology, how they fit in and what the gaps are. It will also include a discussion of some manual approaches which are prime candidates for automation to fill those gaps.

### 3.0 TSD TOOLS DESCRIPTION

#### 3.1 Introduction

During the analysis phase of system design, the emphasis is on the requirements; first, the functional requirements for developing an understanding of the system and the problem it is to solve and, second, the performance requirements to bound the capabilities necessary on a function-by-function basis. Since one of the key premises of TSD is the absolute necessity for requirements traceability and since the methodology is based on developing requirements driven design, it became clear very early that some kind of requirement data base manipulation capability is very necessary. This is due to the often large number of requirements (large programs typically have in the 1000's) and the impossibility of doing tracking, categorization and manipulation manually. In addition, heavy emphasis is placed on requirements definition because of the historical evidence that vague or insufficient requirements often lead to disproportionately high system cost or subsequent design deficiencies.

The second major part of the analysis phase is the functional analysis. Actions during this phase include what is often termed functional decomposition or the top-down partitioning of the system functions into more readily understandable entities. It would be highly useful to have a tool which aids this functional analysis.

The final part of the analysis process is called requirements allocation in Figure 2. Requirements allocation refers to the task of deciding the projected implementation of each functional requirement. This must be done with a full appreciation of the performance requirements of each function and within the context of project specific guidelines for allocation. The key

technical issue in this process is determining the guidelines for allocation. Figure 3 shows an example of such guidelines for one program.

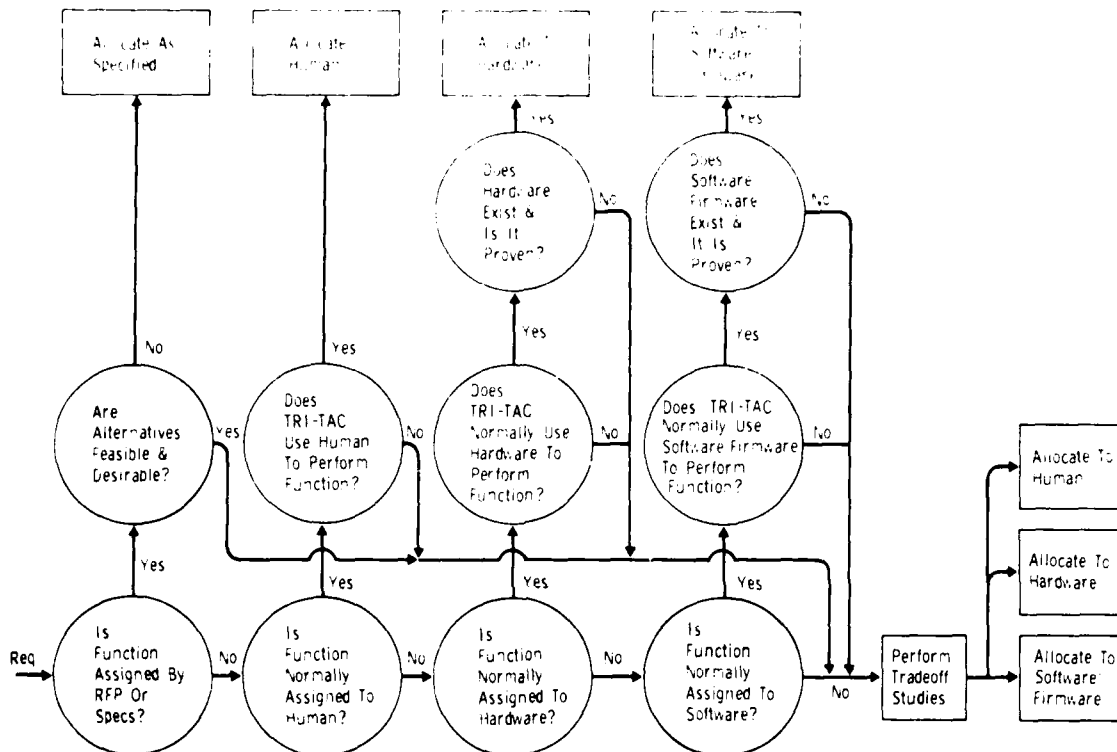


Figure 3 Example Requirement Allocation Guidelines

Figure 3. Example Requirement Allocation Guidelines

From Figure 3, it is obvious that requirement allocation is a manual process. It is a prime candidate for iteration, however, when subsequent validation shows up deficiencies. The key part is the determination of guidelines and the tradeoff studies which define the allocations to be made. There do not exist any tools to aid this task.

For the synthesis phase, major emphasis is on the development of a validated system baseline, the definition of the major hardware, software and people components, and the interactions/interfaces between them. The



process of developing candidate solutions to the functions has two primary approaches. The first is a propose/dispose action where alternative candidates are postulated and then evaluated to attempt to derive a best fit solution. The second is a form of bottom-up design which uses the requirements allocation and proposes compositions of functions to dictate the candidate solutions. These again must be evaluated in terms of how well they solve the problem. Regardless of the approach, the development of candidate systems to solve the problem is a very creative process, depending on the experience, knowledge and background of the system designer. There is currently no way to accomplish this task by automated tools. However, the second half of the task, the evaluation or validation, virtually requires a tool or tools to aid the designer. In the final analysis, there are really only two ways to validate a candidate system design. The first is the prototype method often used in hardware design. The idea behind this is to actually build one to see if it does what is desired. For systems, this is clearly impractical. The second is the simulation method. This consists of the use of a simulation model of the proposed system to answer the relevant questions.

Simulations, by their very nature, are very flexible. They may be developed in a wide range of levels of detail. As the answers required become more specific, the attendant simulation may become more precise with respect to its representation of the system being studied. When the simulation becomes unwieldy, there is the companion technology of emulation to aid the evaluation process. Thus, the key automated tool for the synthesis phase is simulation/emulation for evaluation and validation of candidate system solutions.

With these actions and the applicable support requirements in mind, this section will cover the tools and methods currently in use supporting TSD.

### 3.2 Multi-Level Expression Design System (MED<sub>Sys</sub>)

The Multi-Level Expression Design System (MED<sub>Sys</sub>) is a group of inter-related, automated tools designed to assist an analyst by keeping track of a variety of things as he progresses through the design. Having been conceived and developed by software analysts, MED<sub>Sys</sub> currently retains a strong software flavor in its documentation (such as the statement in the MEDL-R users guide which says that MED<sub>Sys</sub> is "...a systematic method of managing and controlling the software development cycle"). However, the currently existing parts of MED<sub>Sys</sub> have been found to be equally useful for system development through actual use on projects. This result is to be expected, since the typical software development cycle is not fundamentally different from the system development cycle.

The multi-level expression design system is composed of the following components:

- o MEDL-R (Requirements)
- o MEDL-X (Document Generation)
- o MEDL-D (Design)
- o MEDL-P (Procedure/Behavior)

There was specific emphasis in the development of the philosophy of MED<sub>Sys</sub> on separating the tool that supports requirements (MEDL-R) from the tool that supports design (MEDL-D). The reason for this emphasis is the perception of the design process which divides it into the analysis and synthesis phases discussed earlier. The analysis phase deals with the system in such a way that design issues are undesirable and premature. It was found that other require-

ments tools (e.g. PSL/PSA) tended to encourage the user to include design concepts in the requirements phase. Thus, a major effort was put into making a clear distinction between requirements and design and in building tools which will not allow the user to inadvertently cross the boundary between the two.

### 3.2.1 MEDL-R

In terms of implementation, MEDL-R is fully implemented, has been in operational use and is relatively mature. The first version of MEDL-D has been implemented but the use has been mostly test cases to date. MEDL-X and MEDL-P are currently in the design stage. Based on this status, the following discussion will concentrate on MEDL-R. MEDL-R provides the basic requirements tool which is so necessary in the analysis phase and which provides for traceability in the later phases of system design.

The requirements specification phase of the design process requires the translation of needs into statements which specify the functions which must be performed in order to meet those needs.

An important output of the requirements phase should be a detailed, clearly written, functional description of the system and the manner in which the user will interface with that system. Although budgets, resources and schedules are also specified during this phase, it is within the functional specification that most of the errors and difficulties are encountered.

It should come then, as no surprise, that the difference between success and failure on a project may often lie in the clarity, consistency and completeness with which the user's needs have been stated in the requirements document.

Inadequate requirements analysis is quite often manifests itself in four ways:

- o A top-down design is impossible.

- o Adequate testing is difficult or impossible.
- o The user is locked out of the development process.
- o Project management is not in control.

A well-written requirements document will (or should):

- o Be a well-thought-out, complete and non-conflicting record of the user's needs.
- o State specifically the performance of the completed product and the methods which will be used to produce it.
- o Describe the common objectives of the participants.

The MEDL-R processor has been developed and implemented in order to assist the requirements analyst in the production of documents of consistently high quality and reliability.

As the motive behind the design and development of a system, it is imperative that requirements be accurately recorded in a malleable data base. Due to the evolutionary nature of the software development cycle, the MEDL-R processor employs a (malleable) data base that readily lends itself to the changes which are an inevitable by-product of an iterative process.

A more complex problem is that of providing a "mapping" scheme which allows traceability from requirements to modules. The MEDL-R processor addresses this problem via a requirements "taxonomy" through which an attempt is made to identify and classify pertinent characteristics.

MEDL-R addresses the problems of incompleteness and ambiguity by requiring the user to impose successively greater degrees of formalism and clarity upon the problem statement. MEDL-R then, by asking the user to respond to a few "rigorous" questions and by allowing the user to respond to other (less rigorous) questions in a relatively "free-form" manner, gradually leads the

user into imposing discipline upon the design process.

The real-world model upon which MEDL-R is based is really rather simple. Requirements exist and thus may be categorized. This categorization is accomplished via three (primary) aspects:

- o The nature of a requirement,
- o Its motivation, and
- o Its general subject matter.

The nature aspect identifies the most general characteristic(s) of a requirement in terms of how it came about, or how it impacts the system to be designed.

Motivation allows the user to (simply) state the reason(s) why a given requirement is under consideration.

- o The subject aspect allows the user to specify (and thus highlight) the key words of the requirement in the jargon of the target system.

One of the more important objectives of MEDL-R is that it facilitate design as a process and aid in the transition from requirements-definition to a top-level design. This facet is accommodated via a mechanism which is used to describe the resolution of a requirement. The resolution of a requirement is not only a means of further categorization, it carries with it a stronger design connotation and allows a clearer expression of an abstract (as yet) concept. It is a means of identifying design components which will be reduced to:

- o A function
- o A data entity, or
- o A system resource.

As may be seen, MEDL-R relies heavily upon the use of key words to simplify and clarify the representation of requirements. Each MEDL-R key word provides an aspect of characterization or a criterion subset that is associated with a given requirement and implicitly relates common requirements. Thus, all requirements that are of nature procedural would be (implicitly) related.

MEDL-R provides the user with a wide variety of requirement interrelationships which may be used to produce various reports. A requirement may be "decomposed" into logical sections as shown below:

Identification and  
Characteristics

Tracing

Resolution

The identification and characteristics section provides each requirement with a unique number (ID) and allows the user to provide descriptive (and characterizing) information about that requirement.

The tracing section provides the user with the ability to trace the history of a requirement in terms of succession and the decomposition of a complex requirement into smaller, clearer components.

The resolution section contains statements that achieve the "mapping" from the MEDL-R to MEDL-D data base.

MEDL-R is thus the initial software tool of the TSD Methodology. MEDL-R responds to the critical objective of requirements specification and assessment. Specification is enabled by a unique high order language whose elements are processed and retained in a relational data base. Assessment techniques access this data base and produce information about the requirements statement.

The MEDL-R system is intended for use throughout a system development life cycle. It can be used to capture initial requirements, it supports detailed requirements and iterative refinement of constraints, and it supports management control and traceability functions. For high level design, it can be used to help develop the overall design structure. Even during later development phases when detailed design is underway, the MEDL-R scheme is useful for relating requirements to components and checking interfaces.

The retention of requirements in a malleable data base structure is a key factor of this approach. Not only can all requirement interrelationships and dependencies be retained, but all versions, updates, supercessions and obsolete items can be "archived," activated, processed, summarized or analyzed using the features and commands of the MEDL-R support software.

This package is an interactive tool built and used in the environment of a software engineering facility. As such, it is accessed through a terminal by a cognizant requirements analyst, systems designer or manager. Initial creation of the fundamental requirements data base is only a very small part of the requirements definition and assessment task. It is the subsequent refinement, revision, expansion and accommodation of new and changing requirements that has been so burdensome in the past. MEDL-R attacks this problem along with providing evaluation techniques which produce analyst feedback for assessing the impacts brought on by the variable nature of a requirements set.

The MEDL-R software system is composed of several subsystems as shown in Figure 5. The interface package handles general system access and user interaction through a series of directives which determines the subsequent tasks to be done. The six major subsystems of MEDL-R are:

- o CREATE - The Create process is used to generate the requirements source file through a "template" generation process in which the

user fills in a blank area in a template. Figure 4 shows the MEDL-R statement list. The file resulting from the Create process is passed to the Language Processor for creating the actual requirements data base.

REQUIREMENT	SUBJECT
DESCRIPTION	EXPLANATION
NATURE	STATUS
CONSTRAINT	REPLACES
SUBSYSTEM	REPLACED-BY
SCOPE	DERIVES
VERSION	DERIVED-FROM
SOURCE ORIGINATOR	FUNCTION-RESOLUTION
RESULTING-FROM	DATA-RESOLUTION
RESPONSIBILITY	RESOURCE-RESOLUTION

Figure 4. MEDL-R Statement List Prompts

- o UPDATE - Update allows the operator to update the existing system. Update operations include changing any user entered fields of an existing requirement, changing a requirement name or system name, and adding all new requirements to the existing system. The file resulting from the update process is passed to the Language Processor for creating the actual requirements data base.
- o TRANSLATE - The Translator takes a file from the Create or Update process and enters a totally automatic data base build or update process.



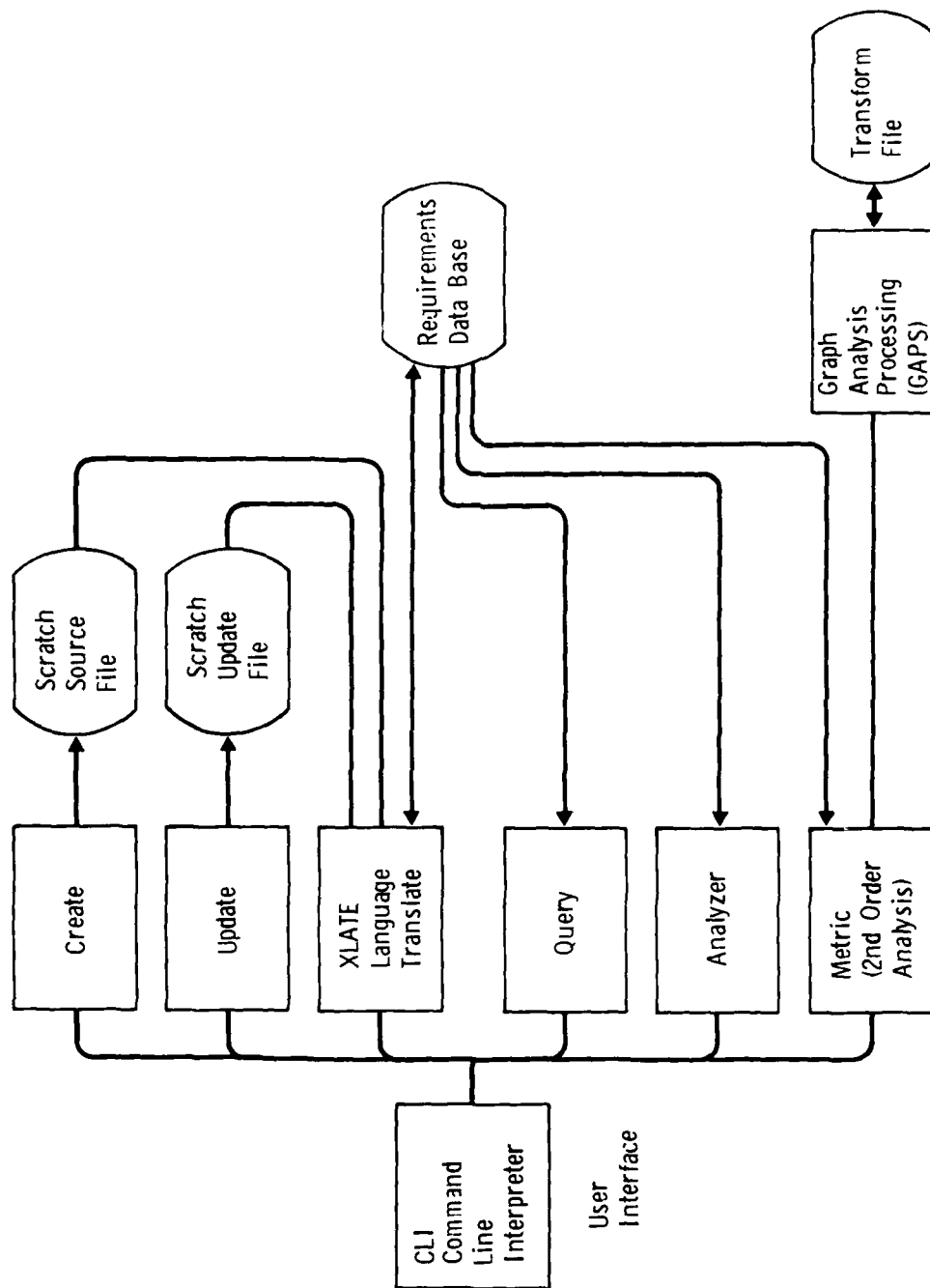


Figure 5 MEDL-R Software System Structure

- o QUERY - The primary purpose of the Query subsystem is to provide the MEDL-R user with a means of extracting from a MEDL-R data base some subset of the information contained. Through this facility, the user can form clusters, eliminating the need to sort through an entire requirements statement to find those requirements that are of interest to a particular user.
- o ANALYZER - The Analyzer subsystem allows the user to list requirements that are in the data base, either singly or the entire requirements expression; to summarize the information in the data base in a tabular format; to build the Formatted Requirements Statement; and to obtain various measurements of what the requirements expression contains.
- o METRIC - The Metric subsystem also provides analytic measures of a data base, but differs from the Analyzer subsystem in that the Metric subsystem executes as a two-step process the Extract transformation routines and the Graph Analysis Processing (GAPS) routines. EXTRACT converts selected requirements data base information into a tree structure of graph nodes and links, which becomes the basis for subsequent analysis. The GAPS package is explained further in another section.

### 3.2.2 MEDL-D

MEDL-D is part of the MED<sub>Sys</sub> and is intended to be used for describing systems design in conjunction with a preceding requirements level (MEDL-R). However, it can be used independently of any of the other MED<sub>Sys</sub> levels.

The link from MEDL-D to MEDL-R occurs through the use of common subsystem, function and data names. MEDL-D contains statements that trace a design component back to the requirement in MEDL-R from which the design was generated. A "complete" requirement in MEDL-R (i.e., one whose resolution is specified because of the amount of detail) provides a link to the design phase (and MEDL-D) because resolution occurs in the following categories:

- o Function resolution - do something.
- o Data resolution - use something.
- o Resource resolution - occupy something.

Design is recognized as an iterative process of imparting successively greater degrees of procedure or behavior on a structure base. The structure is evolved from the requirements during the initial design activity. MEDL-D is primarily oriented to capturing system structure while secondarily oriented to providing the basics of procedure.

The two categories of components in MEDL-D are:

- o Functional objects - considered to have certain characteristics called properties; and
- o Data objects - considered to have certain characteristics called attributes.

Figure 6 shows that functional objects are related to data objects by an "action" which represents the "use" concept. MEDL-D allows hierarchical definitions for functional decomposition and data structuring. The capability to specify discrete scheduling and timing information is contained within the functional specification and is called the functional "control."

### 3.2.3 MEDL-P, MEDL-X

MEDL-P, the "procedure" phase of the Multi-Level Expression Design

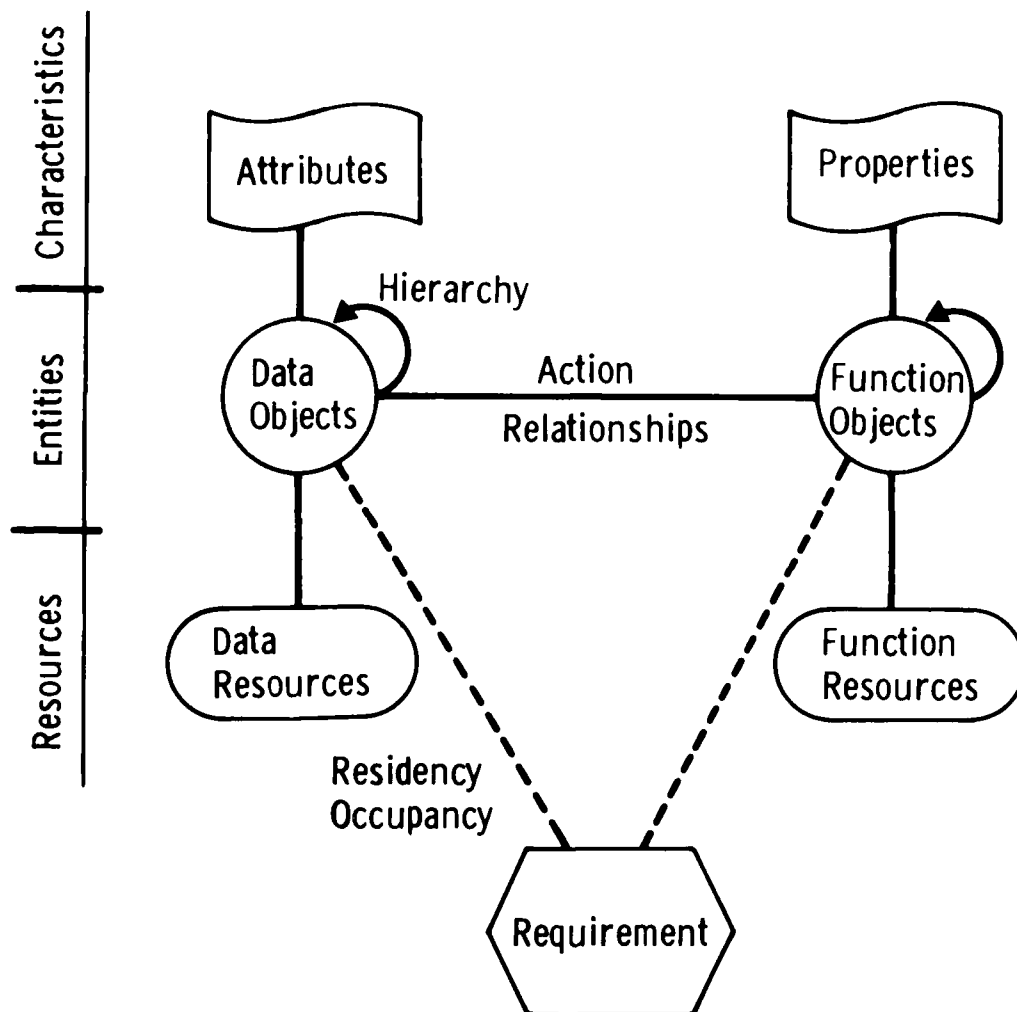


Figure 6 Design Level Model

System and MEDL-X, the document generation phase, are still under development.

#### 3.2.4 GAPS

The Graph Analysis Processing System (GAPS) has been referenced under the Metric subsystem of the MEDL-K software system. To reiterate, the EXTRACT process of the Metric subsystem converts selected requirements data base information into a tree structure of graph nodes and links, which in turn is capable of being analyzed by the GAPS package.

As an overview, a relationship is defined for any two requirements having the same 'NATURE,' 'SUBJECT,' 'MOTIVATION,' or 'FUNCTION,' 'DATA' or 'RESOURCE RESOLUTION. The count of the number of identical relationships that exist between two requirements becomes the link between the two requirements. The link is represented in the GAPS matrix. GAPS has commands to transform the input graph matrix in the following ways:

- o Adjacency Matrix
- o Distance Matrix
- o Sequenced Matrix

Other program features include a clustering algorithm to determine partitions of the sequenced-distance matrix and a command to evaluate a graph partitioning by computing a strength and coupling measure. Another option allows computation of a set of statistics at any point. These statistics include measures of node connectivity and disparity as well as overall graph values for centrality, radius and distance. Commands are also available to print the current matrix and SAVE/RESTORE the current matrix.

GAPS provides a tool to graphically analyze requirements descriptions. Analysis includes an attempt to address the problems of reliability and maintainability of the system by methods which present and emphasize graph

nodes of maximum potential improvement. GAPS can focus its analysis on individual nodes or the total network structure.

### 3.3 Comprehensive Computer Network Modeling (CCNM)

Computer networks of both general-purpose and specialized configurations are, or are becoming, part of every major electronic system being proposed or developed. The requirement for these multicomputer configurations is being generated by both technical and economic pressures. The technical user is requiring greater computing capability applied to broader areas of use with higher reliability. Economy requires that the user spend only what is necessary to solve the immediate problem and only allow for expansion capability that is modular and purchasable when needed. CCNM is an interactive network modeling tool designed to be receptive to iterative design and analysis techniques. Although implementation is not complete, CCNM represents one of the types of simulation tools considered necessary for the system design process.

The philosophy used in assembling the CCNM system is to investigate, detail and implement an experimental network design according to the level structure in Table 1. This level structure provides the capability to model networks at a variety of levels and to increase specific details as more of the characteristics are defined.

When the CCNM system is operational as described, it will continue to evolve as long as new computer systems, line types, networking algorithms (protocols, flow control, routing) and analysis and optimization algorithms come about and are absorbed in the CCNM data base. It is important to point out that, as more networks are modeled, simulated and validated using the CCNM system, the data base of nodes and links will grow rapidly until the most-used processors and standard line types are validated and preserved

to be used by any future network possessing those elements. It is evident that development time for the creation of network simulation models will become drastically reduced, especially at the higher levels.

Table 1. Comprehensive Computer Network Model (CCNM) - Level Structure

Level
1. Node Linkage, Node ID, Line ID, Graphic I/O: Use only standard model available from model library. All variables assumed.
2. Modification of Standard Models: Node, multinode, lines, environment, security, failure options.
3. Alternative Model Call Up: Node, multinode, lines, environment, security, failure actions, control algorithms.
4. Build Models: Language selection, framework with fill in, syntax linkage validator, library access and deletion.
5. Abstract Models (FORTRAN and/or assembly language).
6. Macro Level: Macro encoding, code entrance, data entrance, trace and timekeeping-enabled.
All Interactive or batch linkable to all levels.
NOTE: Each increasing level represents greater model detail.

#### 3.4 Non-Automated Tools/Procedures

There are a number of procedures which also support TSD. These procedures are not automated and serve primarily as documentation tools, allowing the system analyst to express his thoughts and ideas in a form which allows him and others to view and understand them.

Two of the procedures and notations which are notable address the func-

tional definition of the system. They are designed to allow the user to express functional relationships in a graphic way. The two found to be most useful are data flow diagrams as defined by Yourdon, Inc., and N<sup>2</sup> analysis designed by TRW.

#### 3.4.1 Data Flow Diagrams

Data flow diagrams<sup>(1)</sup> (DFD) are used to represent a system pictorially, thus reducing the amount of narrative needed. A DFD is a network representation of a system. The system may be automated, manual or mixed. The DFD portrays the system in terms of its component pieces with all interfaces among the components indicated. A DFD does not represent the flow of control or the order of processing. Numbers used on the diagrams are for identification purposes only. Data flow diagrams are made up of four basic elements:

- (1) Data flows, represented by named vectors, are pipelines through which packets of information of known composition flow.
- (2) Processes, represented by bubbles, are transformations of incoming data flow(s) into outgoing data flow(s). Each process bubble needs a descriptive name.
- (3) Files, represented by two straight horizontal lines, are temporary repositories of data and may consist of tapes, discs, card sets, index files or data bases.
- (4) Data sources and sinks, represented by boxes, are persons or organizations lying outside the context of a system, that are net originators or receivers of system data. A source

---

(1) Tom DeMarco, Structured Analysis and System Specification, New York: Yourdon, 1978.



box exists only to provide commentary about the system's connection to the outside world.

Data flow diagrams are expressed in levels. The first level, called the Context Diagram and shown in Figure 7, portrays an overall picture of the system with four subsystems shown. These are labeled 1 through 4. The subsystems are broken down in separate DFDs and further described as shown in Figure 8. The components of the first subsystem are labeled 1.0, 1.2, 1.3, etc. When a subsystem has been decomposed to as simple a form as necessary, it is called a functional primitive.

There are many advantages to using leveled data flow diagrams. They allow a top-down approach to analysis. By reading the top few levels one can get the big picture or one can begin with the abstract and go to the detailed and narrow in on particular areas of interest. Each page is a complete presentation of the area of work allocated to it. All diagrams can be restricted to 8½ x 11 inch paper.

The second part of the system functional definition consists of the mini-specifications which are concise descriptions of the bottom-level bubbles (functional primitives). Each mini-spec describes rules governing transformation of data flows arriving at the associated primitive into data flows leaving it. An example mini-spec is shown in Table 2.

To augment the data flow diagram, there is an entity called the Data Dictionary. This contains rigorous definitions of all data flow diagram elements such as data flows, components of data flows, files and processes. These definitions relate all data elements through sequence, selection or iteration. The Data Dictionary appears in Table 3.

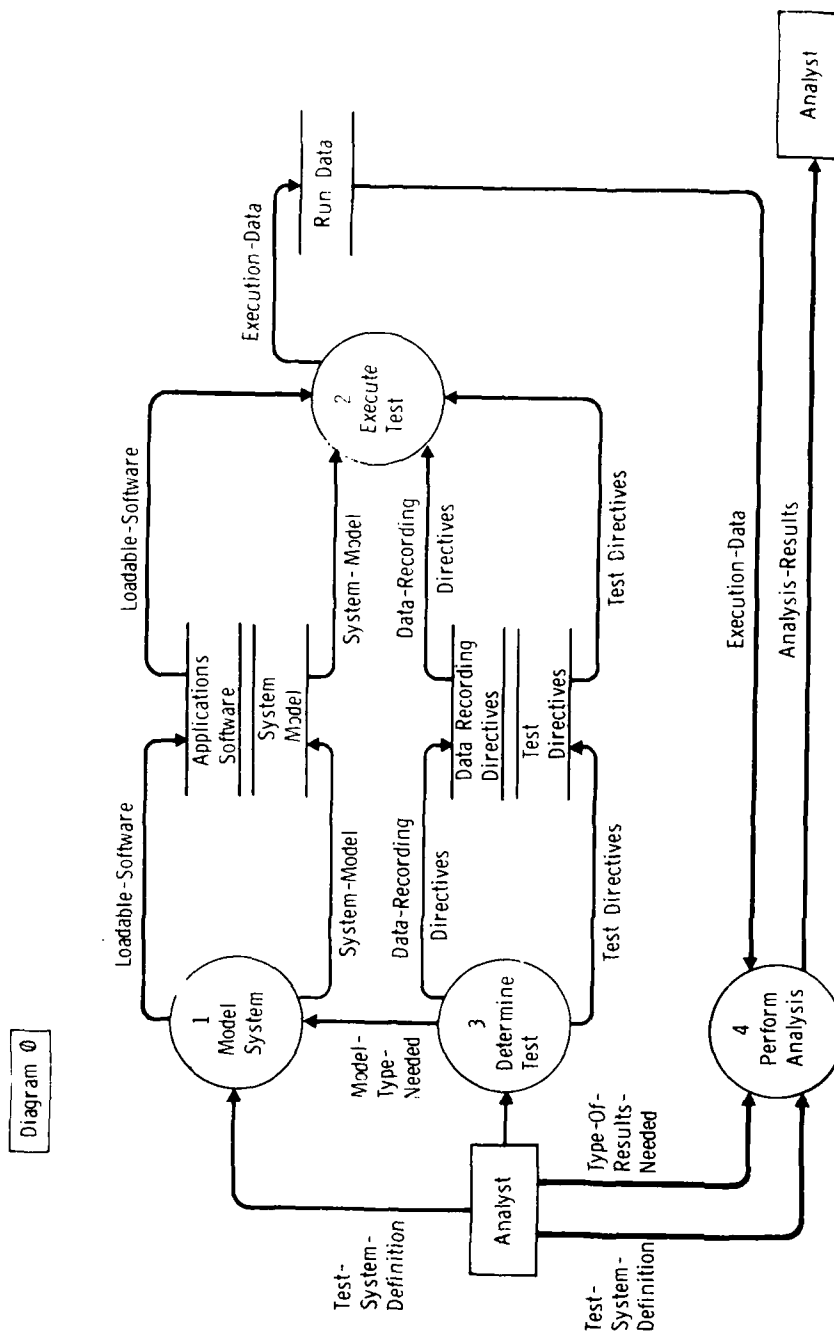


Figure 7 Perform Reliability And Design Analysis

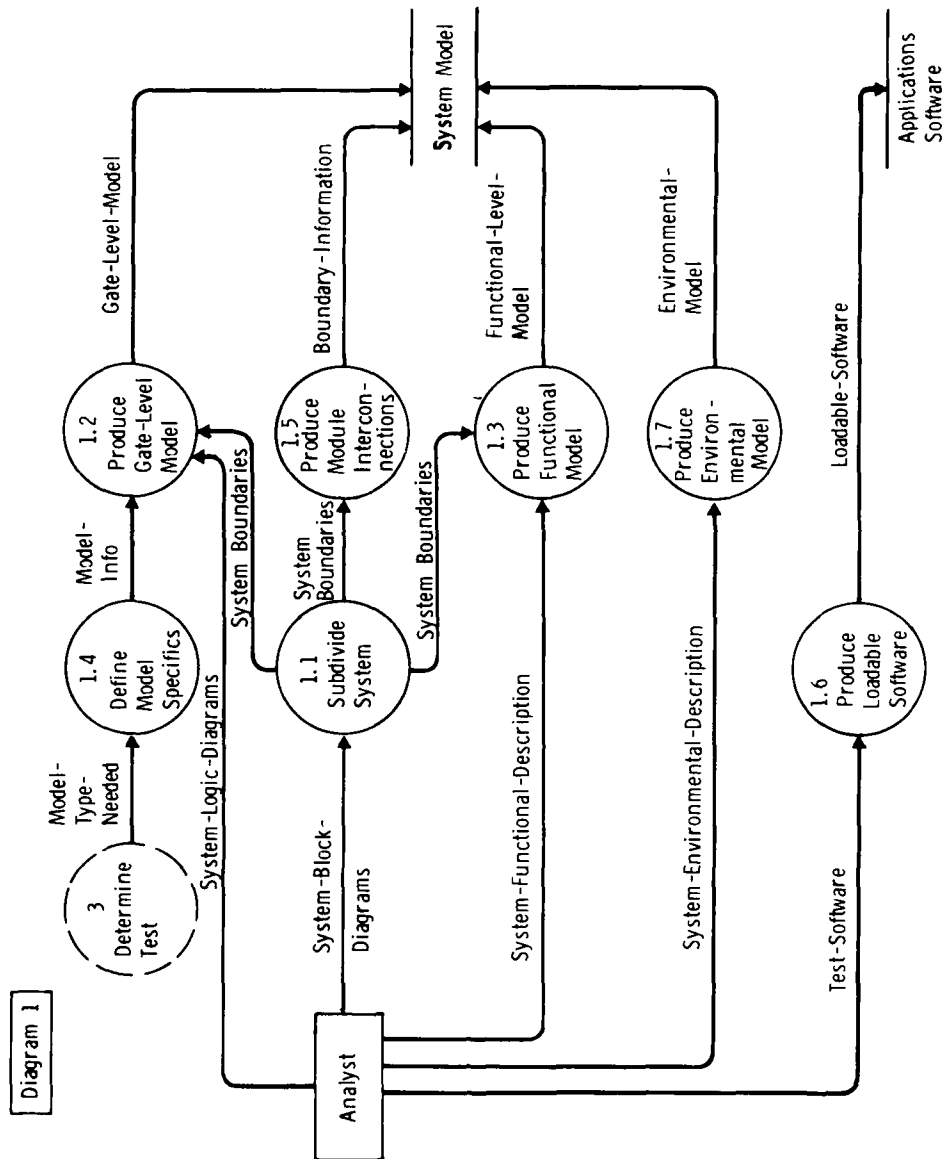


Figure 8 Model System

Table 2. System Modeling Mini-Spec

1.1 Subdivide System

```
IF model-information includes gate-level-model-needed
    THEN
        using system-block-diagram
            partition system into logical functional blocks
            define system-boundaries as internal-interfaces and
                external-interfaces as functional-blocks
    ELSE
        system boundaries are external-interface and functional-block
ENDIF
```

1.2 Produce Gate-Level Model

```
IF model-information is gate-level-model-needed then
    FOR each functional-block in the system-boundaries do
        FROM system-logic-diagrams produce block-gate-model
        TRANSLATE block-gate-model into executable-emulation-code
    ENDDO
    ASSEMBLE all execution-emulation-code into gate-level-model
ENDIF
```

1.3 Produce Functional Model

```
For each functional-block in the system-boundaries do
    From the system-functional-description produce block-functional-
        model and interface-behavior-model
    TRANSLATE block-functional-model to executable-simulation-code
    PRODUCE code-generator-description from system-functional-
```

Table 2 (continued)  
description

```
ENDDO

Assemble all executable-simulation-code into functional-level-model

1.4 Define Model Specifics

IF model-type-needed includes gate-level-model

    THEN

        model-information = gate-level-model-needed and model-
        subdivision-needed

    ELSE

        model-information = monolithic-model-needed

ENDIF

1.5 Produce Model Interconnection

For each of the system-boundaries

    Define boundary-information

1.6 Produce Loadable Software

For each module of test-software do:

    Using the code-generator-description as one input, translate the
    test-software to machine-object-code and generate symbol-tables

ENDDO

Link all machine-object-code modules into loadable-software

1.7 Produce Environmental Model

From the system-environment-description produce environment-model-
description. Translate environment-model-description to
executable-environment-model
```

Table 3. Data Dictionary

( ) optional; [ a | b | e ] alternatives; { } iterations of; + and

analysis results = [ performance-measures | reliability-numbers | failure-effects-results ]

data-desired = [ number-of-samples-necessary + type-data-necessary + type-of failures-desired + { desired-failure-distribution } | type-data-necessary ] + specific-system-portion-of-interest

data-recording-directives = data-to-be-gathered + [ time-interval | time | event ] + output-device + output-format

faults-to-be-inserted = location-of-fault + time-of-fault + duration-of-fault + effect-of-fault

functional-level-model = functional-level-simulation-code + functional-level-symbol-table

gate-level-model = gate-type-tables + gate-interconnection-tables + gate-symbol-table

loadable-software = { machine-object-code + symbol-table }

model-information = [ gate-level-model-needed | model-subdivision-needed | monolithic-model-needed ]

model-type-needed = functional-model-needed + (gate-level-model-needed)

system-boundaries = functional-blocks + { internal-interfaces } + external-interfaces

system-model = environmental-model + functional-level-model + (gate-level-model) + boundary-information

test-directives = { faults-to-be-inserted } + environmental-model-directives + test-conduct-directives

test-system-definition = system-environmental-description + (system-logic-diagrams) = system-functional-description + test-software + system-block-diagram

Table 3 (continued)

type-data-necessary = (gate-performance) + (functional-element-performance) +  
(environmental-model-performance) + (test-driving-factors)

type-results-needed =  $\left[ \begin{array}{l} \text{performance-characteristics} \\ \text{reliability-number} \end{array} \right] \left| \begin{array}{l} \text{failure-effects-analysis} \\ \text{specific-system-portion-of-interest} \end{array} \right|$

### 3.4.2 $N^2$ Charts

The  $N^2$  Chart provides a structured method for the definition of functional interactions and interfaces. The chart itself is a graphical presentation of all of the functions within a system (subsystem, task, etc.), together with the one-way interactions between each of these functions ordered in a fixed coordinate matrix format. The chart gets its name from the fact that for  $N$  functions there are  $N$  squared intersections or squares on the diagram, each of which may contain a function or function interface. The number of possible interfaces for the system is equal to  $N^2 - N$ , where  $N$  is the number of functions within the system. Since both functions and function interfaces occupy a square of the diagram, the total number of squares graphically illustrated is equal to the square of the number of functions involved.

Figure 9 illustrates a simplified  $N^2$  Chart which contains all system functions ( $F_1$  through  $F_4$ ) on the diagonal axis and all system internal functions. The square labeled  $F_1 - F_4$ , for example, represents the one-way interface between function 1 (output) and function 4 (input). All function outputs are defined in the squares which are in the horizontal row of the function, while all function inputs are defined in the squares which are in the vertical column of the function. External inputs and outputs are defined on the top and bottom areas and the side areas respectively. The system illustrated in

Figure 9 has two external inputs (to  $F_1$  and  $F_4$ ) and two external outputs (from  $F_1$  and  $F_4$ ). One of the primary purposes of the  $N^2$  Chart is to indicate where interactions and interfaces do not exist. In Figure 9, the squares below and to the right of function 3 are empty, indicating that there are no interfaces between functions  $F_3$  and  $F_4$ . The table at the bottom of the figure defines the basic rules for the  $N^2$  Chart.

Figure 10 illustrates the "arrow" and "circle arrow" formats which provide a clearer visual presentation of interface direction and flow. These format types have been the most useful in the presentation of higher-level design descriptions to a general audience.

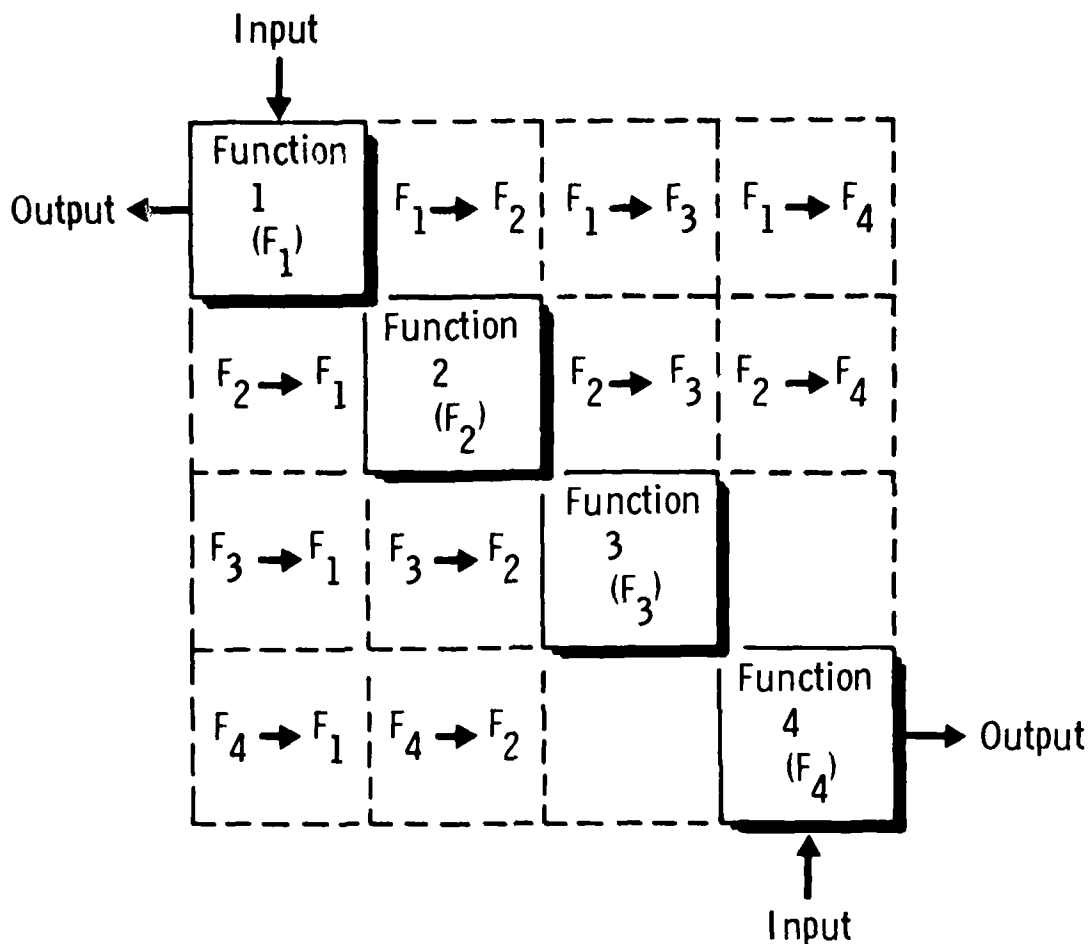
$N^2$  Charts are defined loosely enough to make them useful for a number of different applications. One of the primary benefits in terms of functional analysis is their graphic display of interfaces (and hence some measure of interfunctional coupling). This allows determining reasonable functional grouping as illustrated in Figures 11 and 12.

The  $N^2$  Chart can also be a helpful tool in functionally grouping a given system into easily implementable units or program elements. Figure 12 provides an example of the use of an  $N^2$  Chart to group detailed functions into effectively implementable hardware and/or software elements. Figure 12A shows the result of the detailed functional analysis activity. The dotted lines around the functional groupings of this figure show the initial function collection operation. Figure 12B shows the final implementation functional allocation, which follows the interface minimization grouping of the previous figure.

### 3.5 Other Tools

There are several other tools which support system design but have less general widespread applicability and are hence used on an ad hoc basis.



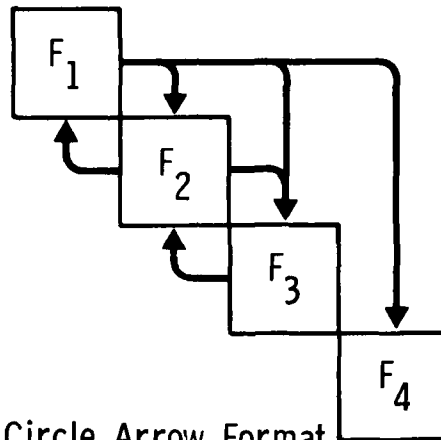


### Basic $N^2$ Chart Rules

- All Functions Are On The Diagonal
- All Outputs Are Horizontal (Left Or Right)
- All Inputs Are Vertical (Up Or Down)
- All Non-Function Squares Define One-Way Interfaces Between The Associated Functions

Figure 9 The  $N^2$  Chart

Line Arrow Format



Circle Arrow Format

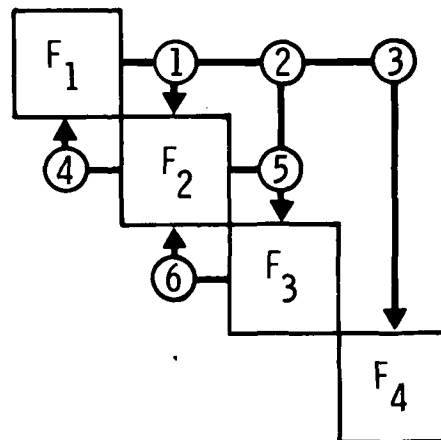


Figure 10  $N^2$  Chart Arrow Formats

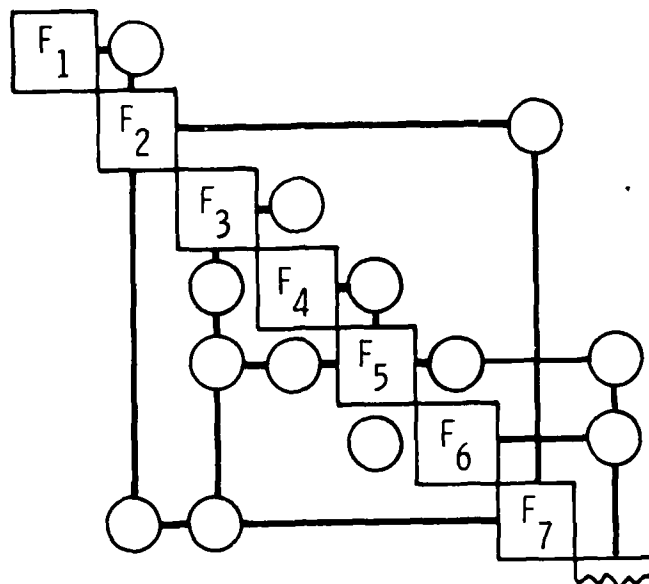


Figure 11A Initial Function Organization

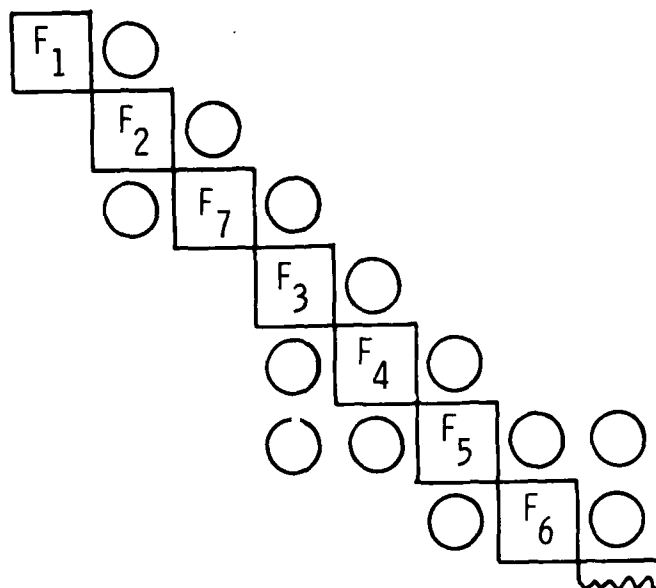


Figure 11B Final Function Organization

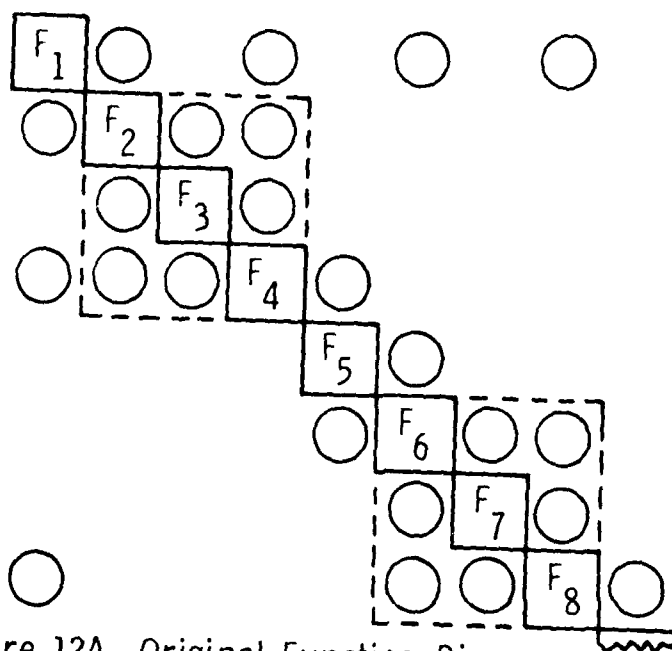


Figure 12A Original Function Diagram

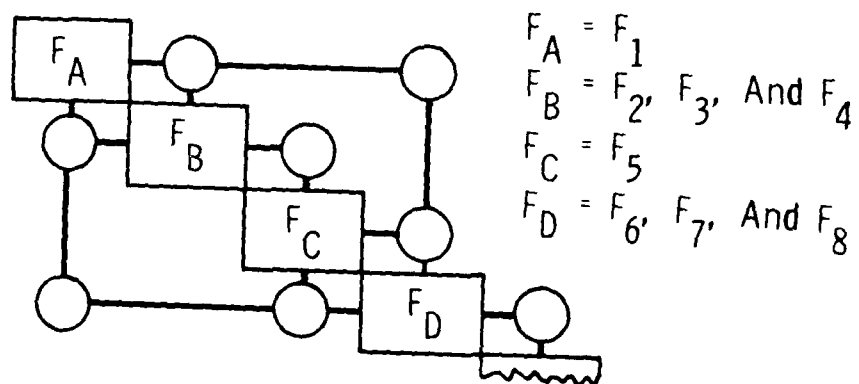


Figure 12B Final Implementation Diagram

Examples of these include simulation to enter specific trade study issues, analytical studies and the use of emulation. Emulation is a special case which bears more detailed explanations concerning its lack of wide applicability.

The key benefit of emulation is that it provides a very detailed model of a piece of hardware with a much smaller amount of overhead than a comparable simulation. This model hence tends to execute faster than a comparable simulation but still retains the flexibility (modifiability) of software. For many systems, however, the issues don't relate to whether the hardware will do the job or not, they relate more to how the various pieces should be connected. There are the network issues addressed by CCNM. Unless very stringent real time considerations are involved or serious form, fit and function constraints are imposed, the detailed hardware/software tradeoffs which emulation can provide are not necessary. Use of emulation also requires very detailed definition of the hardware and software involved (for software, it is the exact code). This amount of detail is often not available until late in the design cycle when the hardware/software trade study issues have already been resolved.

After accepting that emulation is not universally applicable to system design, it is proper to consider when emulation is an absolute necessity in system design. As mentioned before, hardware/software tradeoffs for specific subsets of a real time system require the flexibility of emulation to provide quantitative answers to performance and timing questions. When issues such as weight and power constraints are important, emulation will again provide a performance/capability tradeoff ability. For actual development, emulation is perhaps the only solution when the machine for which development is being carried out is not available or cannot support the peripherals and system software necessary on its own. The major point to be made is that systems

that contain this kind of constraint are only a small part of the total world of systems. In short, emulation is not always useful, but when it is needed, it is often the only viable solution.

There are a number of software tools supporting these other methods. For example, in the area of simulation, there are a number of supported simulation languages such as GPSS, SIMSCRIPT and SIMULA as well as the standard languages like FORTRAN, PL/I and ALGOL which are also usable for developing simulation. The specifics of each simulation, the choice of language and support machine, depend on the problem being addressed and are very much application specific.

For emulation, the support tools are not so prolific. Microcode development tools are emulation support machine dependant and vary from manufacturer to manufacturer. For translating the target software, there are several meta assemblers available. The most used meta assembler at Martin Marietta is the one produced by MacDonnell Douglas for NASA. This meta assembler allows the user to describe the instruction set and formats for the target machine and this description causes the assembler to recognize and translate the assembly language of the described machine.

Since the other tools are not universally applicable to system development, detailed descriptions of them are not included.

#### 4.0 TSD USAGE

##### 4.1 Introduction

While the previous section concentrated on current tools and procedures, it is necessary to emphasize that the most important part of any methodology is the philosophy and concepts behind it and not necessarily the tools supporting the methodology. The tools are important in relation to how well they support the philosophy and how much capability they allow. Tools which by their nature restrict the designer or force him in a direction he doesn't want to take are not only undesirable but potentially dangerous to the success of the design process. Thus, tool selection and use must be accomplished in a judicious manner totally within the context of the underlying philosophy of the design methodology being pursued. With that caveat, this section addresses a system design scenario, identifying the tools which could be used and the points in the process when those tools come into play. Figure 2, presented earlier, will provide the focus for this discussion.

##### 4.2 Proposal

The activities shown on Figure 2 which are prior to the Authorization to Proceed (ATP) represent the initial involvement with the system being designed. This includes the customer's initial development of the system concepts and preparation of the procurement package which begins the process. After receipt of the procurement package, the first major involvement in the process starts with the proposal preparation in response to the Request for Proposal (RFP).

Proposal preparation includes the preliminary analysis of the functions the system is to perform. Due to the very short response times on RFPs, this analysis cannot be to the detail that later analyses will be. However,

within the context of outlining a preliminary definition of what the system should look like, the proposal activity represents a mini system development, leading with an initial system or A-level specification with a preliminary definition of the major hardware and software components proposed. Since the methodology attempts to delay premature decisions concerning the implementation of the system, this is clearly not a logical time to define the hardware and software components. However, the real world situation forces this prematurity due to the current policies concerning system procurement, particularly the proposal evaluation process and the emphasis it places on showing understanding through the proposing of a solution. The real world thus constrains the methodology. The major saving grace of the system is that later stages of the development will revisit the decisions made in the proposal preparation and will either validate those decisions or revise them appropriately.

The tools most useful for proposal preparation are the requirements analysis tools (e.g., MLDL-R), the non-automated tools which aid in the functional decomposition (e.g., Data Flow Diagrams,  $N^2$  Charts) and ad hoc simulation to aid trade studies. The requirements analysis tools allow the manipulation, grouping and categorization of requirements. This is represented by the separation of requirements into functional and other categories so that functional decomposition can focus on what the system is to do without having to consider inappropriate constraints. Through the use of data flow diagrams and/or  $N^2$  charts, the functional requirements are further categorized into major subfunctions and then into more detail until primitive functions are described. Primitive, as used in this context, is a relative term. The extent to which the subdivision process proceeds is dependent on two primary



factors. The first is the judgment of the system analyst that the primitive functions he has developed meet the intended goal of functional decomposition, understandability and realizability. The second factor determining the decomposition stopping point is time. Time is often a factor in proposal efforts and limits the degree of iteration possible.

Once the system has been defined functionally, alternative methods must be developed for performing those functions. At this point the requirements tool becomes important again. The requirements are again categorized, but this time the categorization is by function. This categorization includes not only the functional requirements (which after all caused the functions to be defined) but also the performance requirements and other constraints. This provides a bound on the implementation possibilities of the system. The result of the categorization is a function by function performance definition.

The next step is to determine the inherent parallelism between the functions. This task includes defining what steps must be performed before what other steps, and which steps are independent and may go on in parallel. If  $N^2$  charts were used in the decomposition process, these same charts may be reoriented to show time sequencing. Data flow diagrams, by their nature, assume a parallel system and are not oriented to showing time specifically. Thus, they are not useful for this step. Another manual tool which might aid this process is some variation on Petri Nets.

Once the basic functions of the system are defined, characterized in terms of performance and placed within the overall sequencing pattern of the system, implementation of the functions can be considered. In defining the implementation, the possibility of grouping sequential functions or of multiprogramming parallel functions, of distributing the functions and of providing a single

processor to cover all functions must be traded off against the other general requirements such as cost, reliability, maintainability, power consumption, size, weight, etc. For these trade studies, ad hoc simulations, paper and computer based analyses (for reliability, for example) and engineering judgment will develop one potential, viable candidate solution which will then be documented in the proposal.

The requirements, the functional decomposition and allocation of requirements to those functions and the candidate solution will make up the proposal and the proposed system level or A-level specification.

#### 4.3 Functional Baseline

After contract award and the Authorization to Proceed, the primary emphasis is on developing a system functional baseline or a logical model of what the system is to do. The effort expended in the proposal preparation will serve to provide a starting point for this process. However, during this phase of the contract, more time and effort will go into developing a validated functional baseline. The useful tools for this process, which represents the analysis phase described in Section 2, are the same tools which were useful in the proposal development stage.

The first step, automation of requirements or the entering of the requirements into a data base, has already been done in the proposal effort. The requirement changes, additions and deletions which always develop are added in at this point. In addition, the requirement derivation process, understanding and documenting the requirement level consequences and implications of the imposed requirements, has begun in earnest. The requirements set is continually reviewed and analyzed, often using automated metric tools, for completeness and consistency. Gaps which are not fillable by derived require-

ments are filled by assumed requirements which must be noted and coordinated with the customer. This requirements analysis process continues through the development effort and provides a viable mechanism for traceability and study of ramifications of requirement changes. Requirements tools, such as MEDL-R, will aid immensely in this effort.

The emphasis put on requirements at this stage is not surprising since the methodology is intended to develop system solutions based on the system requirements. It is very important to ensure requirement traceability to accommodate requirement changes later in the cycle.

Concurrent with the general requirement analysis described above, the functional decomposition process described briefly in the proposal section is accomplished. This process deals exclusively with the functional system requirements, those imposed and derived and those developed as a result of the mission analysis. Mission analysis provides details concerning the use of the system in an operational environment. The operations concept should be a part of the imposed requirements given in the statement of work or system specification but there is a possibility that the analysis of the system mission will result in some new functional and performance requirements. This analysis also serves to validate the imposed and derived requirements in terms of necessity to accomplish the system's operational purpose. This could be thought of as assigning a kind of priority to requirements, giving those that don't contribute to mission accomplishment a lower priority than those that do contribute.

The functional analysis consists of the functional decomposition process described in the proposal section. This process is aided by manual techniques such as data flow diagrams or  $N^2$  charts. Since iteration is now a much

more viable possibility, effort in quantifying the quality of the partitioning will prove beneficial. Tools such as GAPS (described briefly in Section 4.2) which aid in the quantification are very helpful. In terms of measuring the result of functional decomposition, concepts such as strength and coupling will allow a number to be assigned to a given partitioning.

Strength, or cohesion as it is often called, represents the intrafunctional interfaces, how much the various portions of a function belong together. From an idealized point of view, the best partitioning results in consistent functions whose components are grouped together because of functional (rather than logical, temporal or coincidental) reasons. That is, each of the component pieces contributes to the one function rather than having been grouped in a "functional" block because they do the same sort of thing but slightly differently (logical) or they do a set of things in the same time period (temporal, initialization is a good example) or they are just grouped for no apparent reason (coincidental). Strength may be somewhat difficult to quantify but a good first cut is to assign a value which is inversely proportioned to the number of different things a given functional block does.

Coupling, or the interfunctional interfaces, is more easily quantified. Coupling represents the complexity or number of interfaces with other functions. The desire is to minimize coupling so that each function becomes independent. This corresponds to the software concept of information hiding.

To analyze the "goodness" of a given partitioning against other partitionings, both the strength and coupling must be calculated. Since the object is to maximize strength and minimize the coupling, the ratio or strength/coupling gives a quantitative indication of how good a functional partitioning is. The better partitioning will have a higher number than a poorer one. Alternative

partitionings are proposed until the analyst is comfortable with one and the assessment process indicates that it is as good or better than the others proposed.

Once a functional decomposition is arrived at, the performance requirements are assigned to each function. The functional requirements are also assigned to the functions but this happens as the decomposition proceeds since the functional requirements dictate the decomposition components. The result of this process, which is accomplished using a requirements tool, is the system functional baseline. The functional baseline describes the basic functions of the system, the interfaces between functions and the constraints in terms of performance for each function. One additional step is accomplished prior to the System Requirements Review, which marks the end of the Analysis Phase. This step is the allocation of the various functions to hardware, software and human elements. As mentioned before, this allocation is project-dependent but it serves to bound the extent of automation within the system. This allocation is aided by guidelines and trade studies to provide the most logical breakdown.

#### 4.4 Allocated Baseline

The allocated baseline represents the top-level system definition. Development of the allocated baseline is driven by the functional baseline, including the performance constraints. At this stage, potential solutions are proposed and evaluated. It is significant that the consideration of physical solutions is not undertaken until after the System Requirements Review (SRR) during which the system functional baseline is presented and agreement by both sides is sought on the total system requirements. Development of the allocated baseline and the subsystem design which follows it represent the

synthesis phase of the development.

The definition of candidate solutions to the functional baseline is currently highly dependent on the experience and background of the system designer. There are no tools which will take the functional model, constrained by the performance requirements, and magically produce the proper mix of hardware, software and human elements to accomplish those functions. However, there are a number of tools which will aid in evaluating proposed candidates to determine their fitness for the problem. This section will therefore concentrate on the evaluation process rather than the system solution proposal process.

There are several ways of guiding the development of potential solutions. The first are the imposed system constraints. For example, requirements for reliability, survivability, availability, fault tolerance, ruggedization, size, power consumption, weight, etc., will serve to bound the potential solutions. The second aid to the system designer is the determination of parallelism in the problem. This determination is made by examining the system functions and determining the sequencing and order independence of each of the functions.  $N^2$  charts are adaptable enough to aid this process and show the order of execution of functions. Data flow diagrams, which assume parallelism of the functions and do not provide a mechanism for describing sequencing or control, are not useful for this task. Variations on Petri nets, with their strong emphasis on sequencing, are another possible tool to aid this analysis.

Regardless of the specific tools used to bound the solution space, the next step is to propose and evaluate potential solutions. The evaluation process is aided in a number of ways by manual and automated tools. For manual evaluation, sizing and timing studies, which seek to characterize the

functions in terms of computer resource requirements, will provide indications concerning which functions may be performed in a single processor, for example. Once potential candidates are defined, simulation of the candidates will provide quantitative data on the performance of the candidates. There are several approaches to this simulation. These include ad hoc simulations, built on a case-by-case basis, and use of general purpose structures such as the CCNM tool previously described.

Other tradeoffs will affect the proposed solution. These include considerations of life cycle cost, risk and logistic support. For components which are critical, or for which there is some concern relating to performance, the use of emulation in conjunction with simulation will provide a detailed examination of capability. For emulation, if the emulation is intended to represent a processor, the software driving the emulation must be developed.

Once all the factors are taken into consideration and the evaluations have been done, one candidate system will emerge as the best of those considered. After assigning the functions (and hence the requirements) to the component pieces of proposed solution, the result is the allocated baseline. The assignment of the functions to components extends the requirement traceability of the methodology so that one may go from the basic requirement to the actual implementation due to that requirement.

The allocated baseline is reviewed and agreed to at the System Design Review (SDR). Following the SDR, further definition is done on the various subsystems. This definition includes the specification of test requirements and further detailing of the subsystems. The results of this process are reviewed at the Preliminary Design Review (PDR) which ends the synthesis phase of the development. From here, the system goes into the implementation phase which is not specifically covered in this document about TSD.

## 5.0 GPS USE OF TSD

### 5.1 Introduction

The Navstar Global Positioning System (GPS) is a space-based radio navigation system that provides precise time and three-dimensional position and velocity to GPS users. The GPS will consist of three major segments: the Space Segment, the User Segment and the Control Segment. TSD methodology was applied to the system definition of the operational Control Segment of GPS. GPS represented an ideal test case since the solution was not predefined by the customer and true top-down design could be done.

This discussion is not intended to be a full explanation of the GPS system or even a top-level discussion of GPS itself. This discussion is instead intended to describe the use of the TSD methodology and the lessons learned and to provide suggestions for the future. Thus there will be very few details about GPS. The aspects of what was done in terms of approach will provide the major focus. The only aspect of GPS which is relevant is the fact that it represents a real world case, it is a large system and it contains real time and command and control aspects.

### 5.2 Approach

TSD methodology was used for the Stage 1 contract for GPS which resulted in the B-level specification of the GPS operational control segment. Since development to B-level specifications is the primary focus of TSD, the experiences encountered in applying TSD to this contract are directly applicable to the methodology development.

Due to the fact that GPS was a real contract and not just the experimental application of a methodology to a "toy" problem, the effort was not able to take advantage of any of the tools which were in the development stage (i.e.,



prerelease prototypes). MEDL-R and CONM both fit into this category at the time the GPS project was being done, although MEDL-R has now reached the status of a released tool with version and change control. However, the key concepts of TSD were assiduously applied to the whole development. The primary steps of the methodology were done in the proper order. Particular attention was paid to requirements traceability, functional decomposition and resource allocation. Requirements traceability was enforced by manual methods. Functional decomposition was aided by using N<sup>2</sup> Charts and preliminary system definition was aided by extensive use of simulation.

The requirements analysis phase had four major objectives:

1. Validate the imposed minimum requirements.
2. Determine cost/benefit of the customer identified enhancements.
3. Interpret the imposed growth requirements.
4. Derive lower level detailed requirements.

The validation of the imposed minimum requirements focused on assuring that the requirements satisfied the criteria of being unambiguous, necessary and testable. The customer identified enhancements were prioritized based on mission criticality and were incorporated into the baseline as fixed acquisition cost permitted. The growth requirements were interpreted to define specific hardware and software design requirements. The net result of the requirements analysis included a requirements baseline including all imposed minimum and growth requirements, plus 75% of the desired enhancements.

Functional decomposition was undertaken in several steps. The operational control segment was first divided up into four major components: the ground antenna (GA), the monitor stations (MS), the master control station (MCS), and communications. Although this division is primarily geographical

(there will be several monitor stations and ground antenna which are geographically separated from the single master control station), the functions performed at each point are in fact different and the breakdown is thus also functional. The total requirements were then allocated to each major subsystem to allow the functional decomposition to proceed within each subsystem independently.

The system was further decomposed within each subsystem and then the interfaces between these level 1 functions were analyzed and graphically illustrated using an  $N^2$  Chart. The  $N^2$  Chart proved to be doubly useful due to the fact that the program included three major subcontractors and the  $N^2$  Chart was also able to show the interfaces between companies.

The level 1 functions became the major configuration items (CIs) and computer program configuration items (CPCIs) in the B-level specification. To facilitate the resource definition process, these major functions were decomposed one level further. Requirements on a per function basis were then allocated to the various components (hardware, software, human elements) and served to provide a functional baseline, documented in an A-level specification.

Development of a system architecture to support the functional decomposition used a number of methods to define the major components. Analysis of activity threads, time lines and coupling lead to sizing and timing constraints. Consideration of various modes of processing such as real time, near real time and batch processing were evaluated for each function. Other criteria such as growth and life cycle cost were applied to candidate architecture to determine configuration sensitivities. Extensive design validation, using benchmarking and vendor written software, in parallel with simulation and modeling, proved the credibility of the ultimate architecture.

Following the analysis of requirements and definition of the basic system, B-level specifications were produced. Efforts were made to insure traceability of the requirements to the B-level specifications. The final B-level specification was one of the primary deliverable items under the contract.

### 5.3 Results

The overall results attained on the GPS contract were outstanding. That is not to say that there were no problems during the program. The methodology, while aiding in some cases, hindered progress in others. It also pointed out some failings in government system development regulations which will be covered later. In terms of a test case for methodology assessment, GPS provided a wealth of experience which could be obtained no other way.

The requirements analysis effort suffered the most in the assessment. One of the most difficult aspects of requirements analysis was the effort to provide traceability manually. This effort was successful for the A-level specification and there is a demonstrable traceability from specific requirements to A-level specification paragraphs. Traceability to the B-level was somewhat less successful. The major problem is the fact that B-level specifications are more detailed and hence the number of paragraphs is much greater than for the A-level specifications. In addition, insuring requirement traceability doesn't seem to provide any positive benefit from the point of view of the person tasked to do the job. This barrier requires an education process and effective personal discipline for the lower-level designers. The effort would also have been greatly aided by the use of an automated tool.

The second aspect of requirements analysis that didn't follow the ideal pattern was the requirements allocation, the assigning of requirements (and functions) to hardware, software and human elements. Theoretically, this

allocation provides the basis for composing the basic elements of the system. In practice, it didn't work out that way. Extensive effort was put into the allocation process and a large amount of documentation resulted. This documentation was then never used for anything further. The primary benefit of the effort was the education and enlightenment of the personnel involved, but that was never the intended goal. Automation and a clear definition of how the results of the effort are to be used would have alleviated the problem.

For the system architecture definition, the ISD approach was very successful. The simulation and modeling validation provided for a highly credible design. The one failing in this area was the lack of adequate examination of man-machine interface issues.

Production of the B-level specification produced some specific recommendations concerning specification trees and the requirements of MIL-STD-48 (Configuration Management Practices for Systems, Equipment, Munitions, and Computer Program) and MIL-STD-490 (Specification Practices). The general consensus was that these standards do not provide sufficient flexibility to adequately describe a computer-based system. There needs to be a B0-level specification which bridges the gap between the A-level specification and the B-5 specification. Another suggested approach was to make the B-5 specification subservient to the B-1 specification.

#### 5.4 Conclusions

In terms of use of ISD methodology, GPS provided some very significant conclusions.

1. The requirements analysis must be aided by some sort of automated tool. The front end education costs will be more than paid back in later stages by the fact that the requirements are automated.

2. The use of simulation. The use of simulation is mandatory for the architecture studies phase. It provides an interaction and validation to the design effort.
3. There is not yet a well-defined capability to go from the functional decomposition with performance requirements allocated to the function to the resources necessary to implement those functions. The preliminary design effort is still largely a matter of the experience and expertise of the system personnel. This is a gap in the methodology that needs to be filled by research.
4. The people on a program using TSD must understand it, know how to apply it and believe that it will aid their work. This should come as the methodology is applied to more projects.

Overall, the top-down, structured approach benefitted the GPS project and enabled production of a viable, defensible design.

## 6.0 CONCLUSIONS

There are many descriptions of what should be done during the system design process. Unfortunately, these descriptions concentrate on what rather than how and as a result are only useful in providing a basis for a methodology. A methodology must contain much more than just philosophy. A methodology must contain guidelines, procedures and practices, all of which are supported by either manual or automated tools. It must thus concentrate on the how of system design to be useful at all.

In light of this requirement on methodologies, the TSD methodology described in this report is truly embryonic. Much is still left up to the capabilities of the system designer. Perhaps the biggest gap is the step from understanding what the system must do (functional baseline) to the design which will implement those functions. The process does not flow logically and there are no tools currently available to support this step.

Another problem with the current methodology has to do with interoperability of tools. The tools are not built around common data bases and there is no capability to automatically move from one to the next. There should be no surprise that this is the case due to the fact that currently existing tools have been developed independently. Major work remains to be done in consolidating and coordinating various tools.

Application of the methodology has defined other shortcomings, specifically in the tools. MEDL-R has some capabilities which are not beneficial for system design and lacks some which are necessary. MEDL-D is currently heavily oriented toward software design, not system design. The thrust of these tools is being redirected to make them more responsive.

Data flow diagrams don't provide the capability to show control sequence

the design and this limits their usefulness in expressing the inherent parallelism of the functions.  $N^2$  charts, on the other hand, can be used to show this parallelism, but they have the problem of becoming very large as the number of functions increases. Their size thus limits the willingness of the system designer to change them as the design progresses. Automation might alleviate this failing somewhat.

Simulation suffers from the cost of developing and executing it. Development cost is aided by generalized support capabilities such as CCNM. Emulation requires fairly detailed definition of the portion being emulated and is thus most useful in the more detailed design process.

Regardless of the failings of current tools, the definition of a viable approach to system design has provided a baseline with which to evaluate potential tools and techniques. Further work in the tool area, specifically concentrating on automated tools, is continuing. Martin Marietta is convinced the system design process is capable of being made and must be made more rigorous, traceable and amenable to verification.

☆U.S. GOVERNMENT PRINTING OFFICE: 1981-714-025/83



## *MISSION of Rome Air Development Center*

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C<sup>3</sup>I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.



**DATE  
FILMED**

**3-8**